Computer Graphics

Course outline

- Introduction
- Images and display techniques
  - Bases
  - Gamma correction
  - Aliasing and techniques to remedy
  - Storage

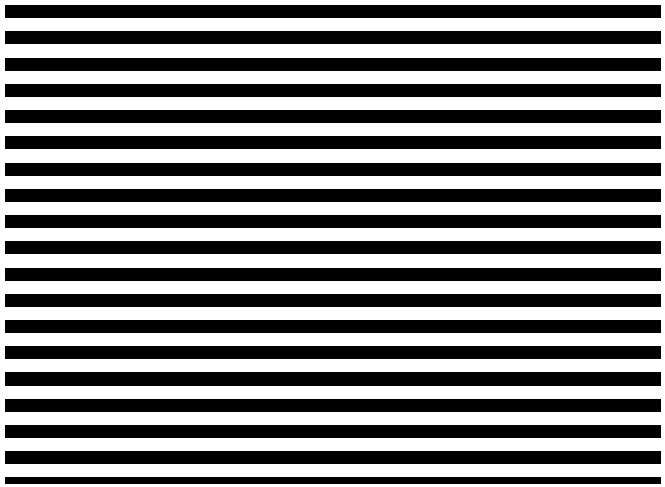
Course outline

- 3D Perspective & 2D / 3D transformations
  - Go from a 3D space to a 2D display device
- Two paradigms for image synthesis
- Representation of curves and surfaces
  - Splines & co.
  - Meshes
- Realistic rendering by ray tracing
  - Concepts and theoretical bases

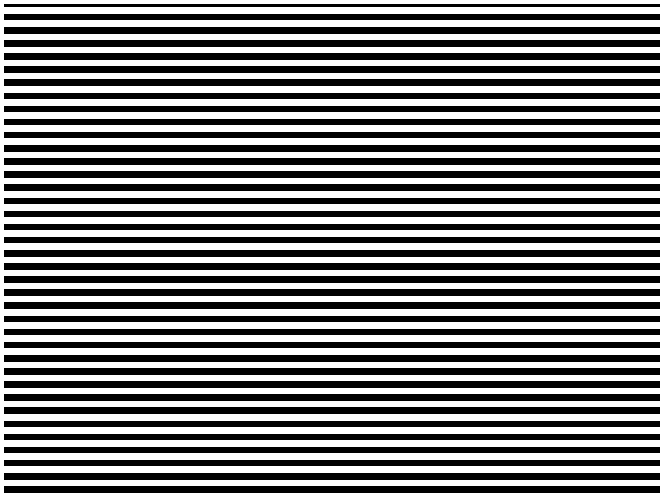
Aliasing

« Aliasing » (crénelage in French)

- Appears during image synthesis or capture, during spatial discretization (transformation of a “continuous” image to discrete pixels)
  - Small experiment : The test image is a series of black and white lines, with an increasing density.



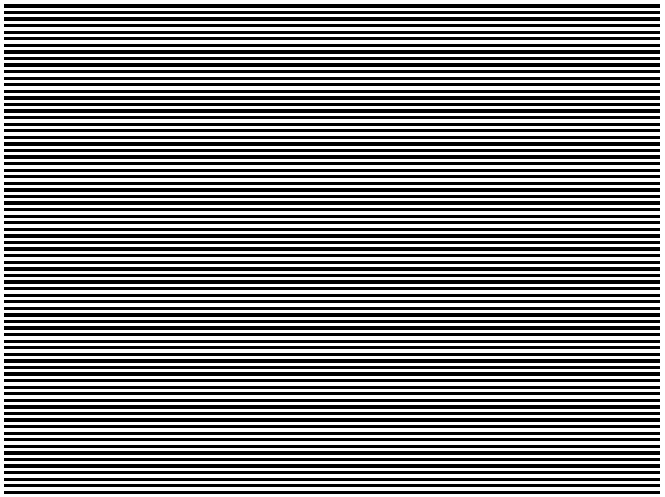
1 line/4 pixels



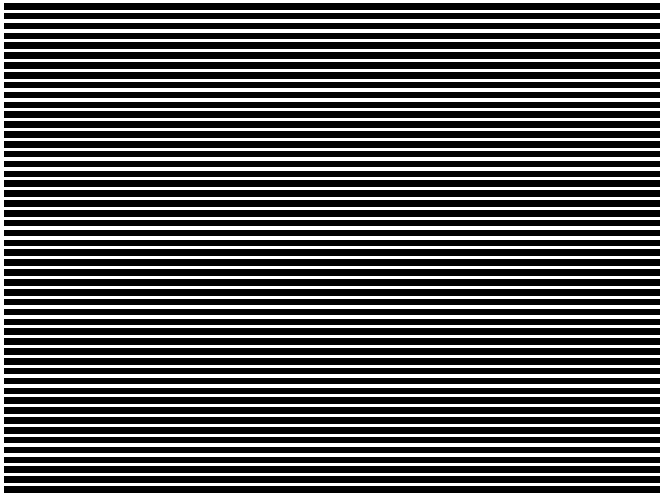
1/2

Computer Graphics

Aliasing



1/1



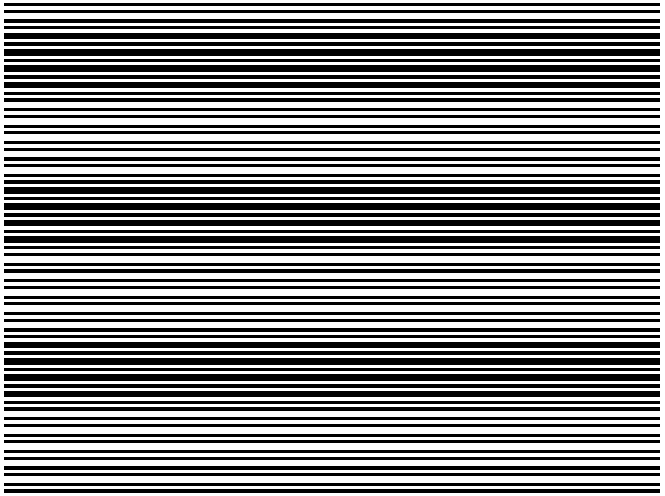
3/2

Aliasing

2/1

199/100

Aliasing



16/5

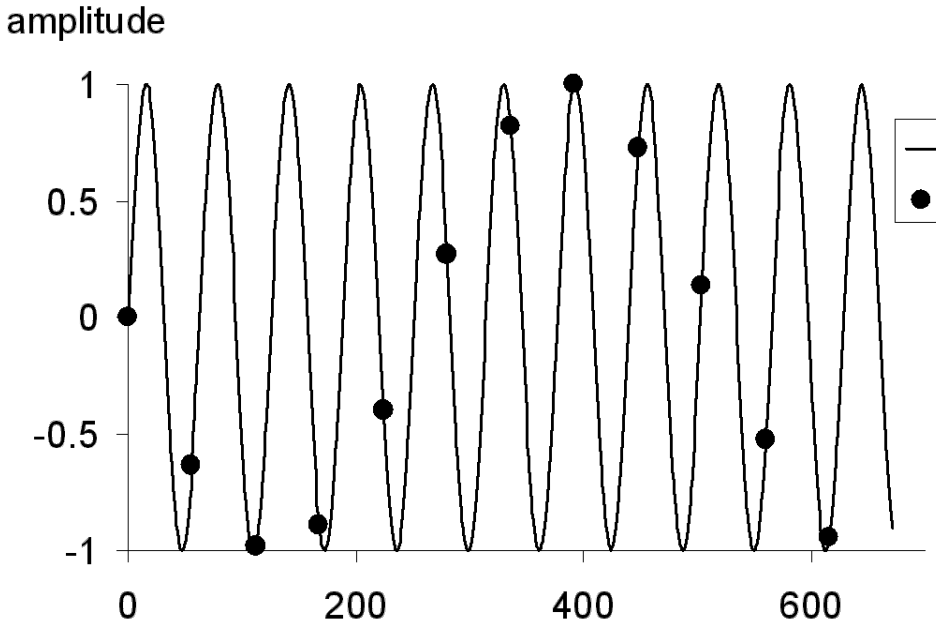
- In fact, one should not have any spatial frequency that is higher than a given cutoff frequency, that depends on the sampling density.

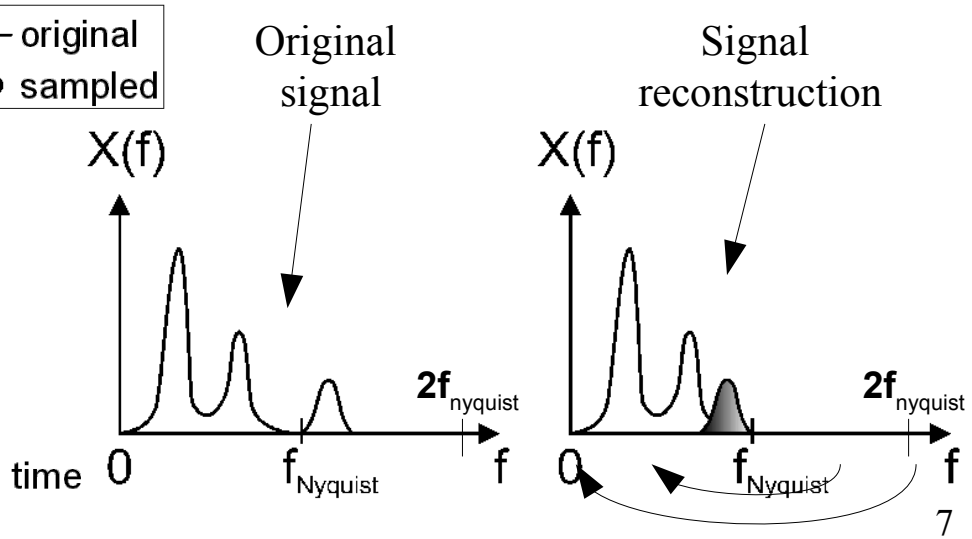
Aliasing

- Nyquist-Shannon theorem
  - If the spectrum of a function doesn't contain frequencies higher than e.g. B, then it is completely determined by a series of samples (in time, space …) separated by 1 / (2B) , or of frequency equal to 2B . 2B is the Nyquist frequency.
- Otherwise, there is aliasing:

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t)\, e^{i\omega t}\, dt$$

Fourrier transform (amplitude)

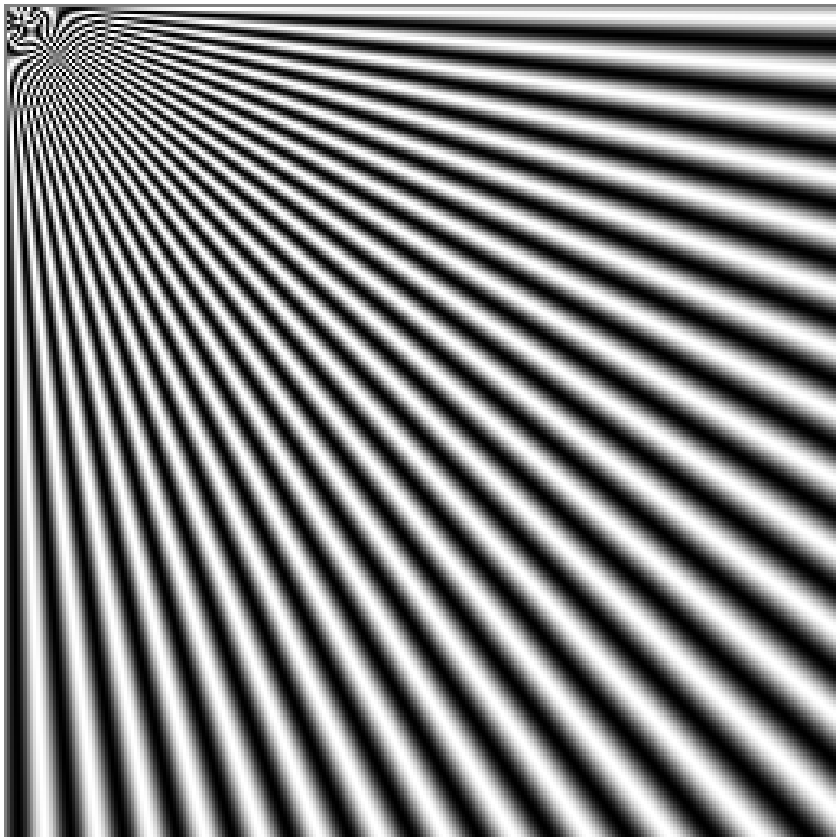


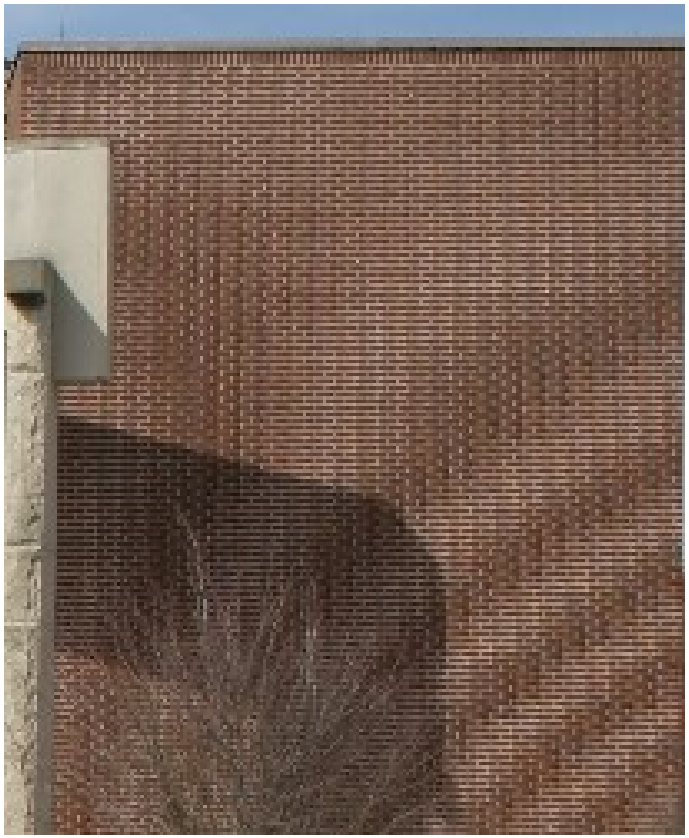


Computer Graphics

Aliasing

- Moiré patterns





Aliasing

How to limit aliasing ?

- By increasing the sampling density to respect Nyquist-Shannon's theorem ?
- By filtering the signal with a low pass filter to be in the window of Shannon's theorem for a given sampling density ?
- A combination of both... ?

Aliasing

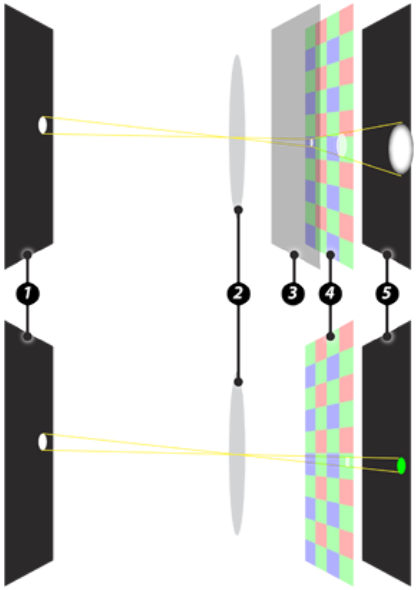
- Increase the sampling density (or rate)
  - Equivalent to increase the image resolution
  - But : real life images are often fractal
    - High frequency details are such that spatial resolution is huge
  - But : artificially generated images include sudden changes in intensity (e.g. black lines on white background)
    - Cf Fourier decomposition of a square signal :

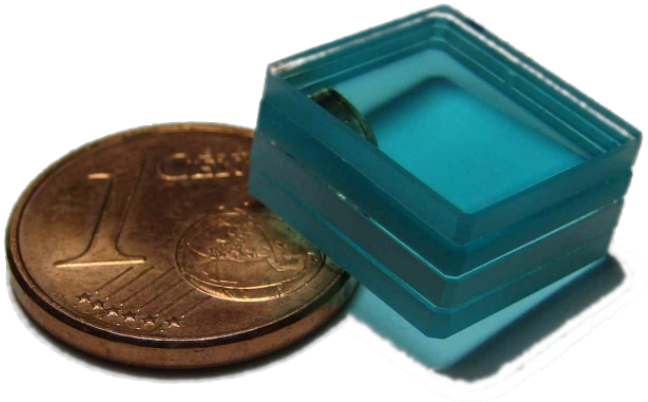
      Spectrum of a square signal : odd harmonics of amplitude 1/n ...

This solution alone is not working well.

Aliasing

- Analog lowpass filter
  - For capturing devices, this must be done before the actual capture
  - Makes slightly blurred image - but not too much!
  - This is exactly what we find on the sensor of digital cameras : An AA filter positioned between the lens, and the sensor.
  - Many technologies are available.

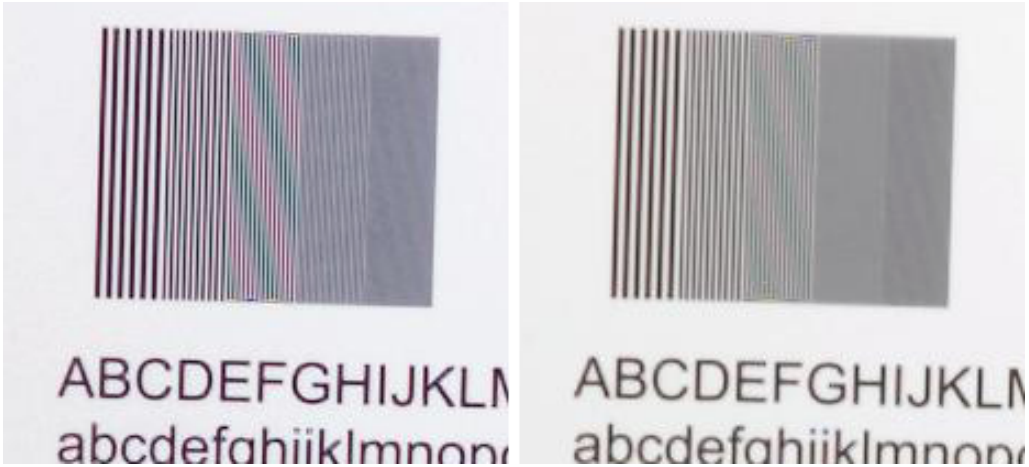




Aliasing

- Example: analog filter (physical) in a digital camera

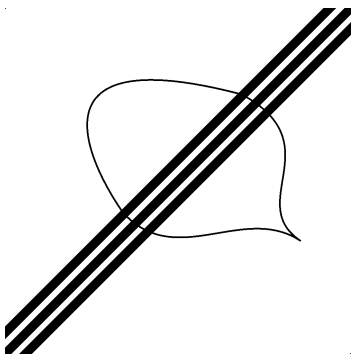
Without antialiasing filter



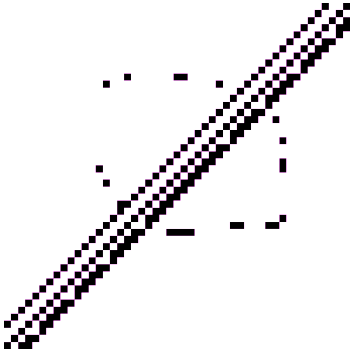
With AA filter in front of the sensor

Aliasing

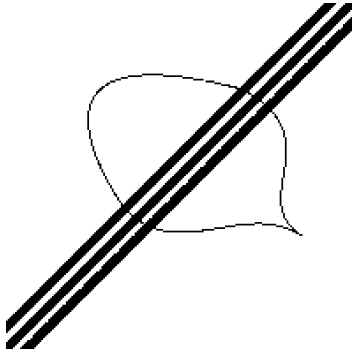
- Numerical low-pass filtering
  - When one creates artificial images, no analog AA is possible
  - The usual way is to use oversampling (sampling with a higher rate), followed by a arithmetic mean to return to the actual (desired) sampling rate
  - It is also possible to introduce variability in the sampling positions (e.g. add a random contribution to the positions of sample in a pixel)



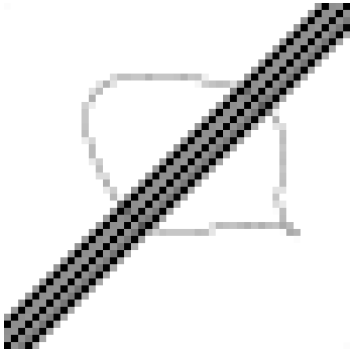
Original image (continuous)



49x49 resolution W/O antialiasing



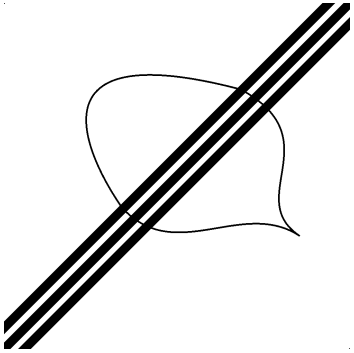
196x196 resolution W/O antialiasing



49x49 resolution with 4x4 oversampling

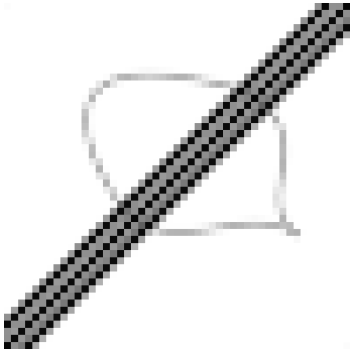
Aliasing

- Simulation of an analog low-pass filter

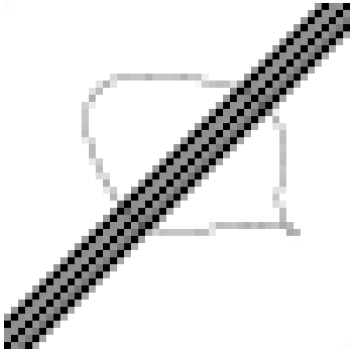


Original image (continuous)

"Blurred" original image



49x49 resolution from the "blurred" image



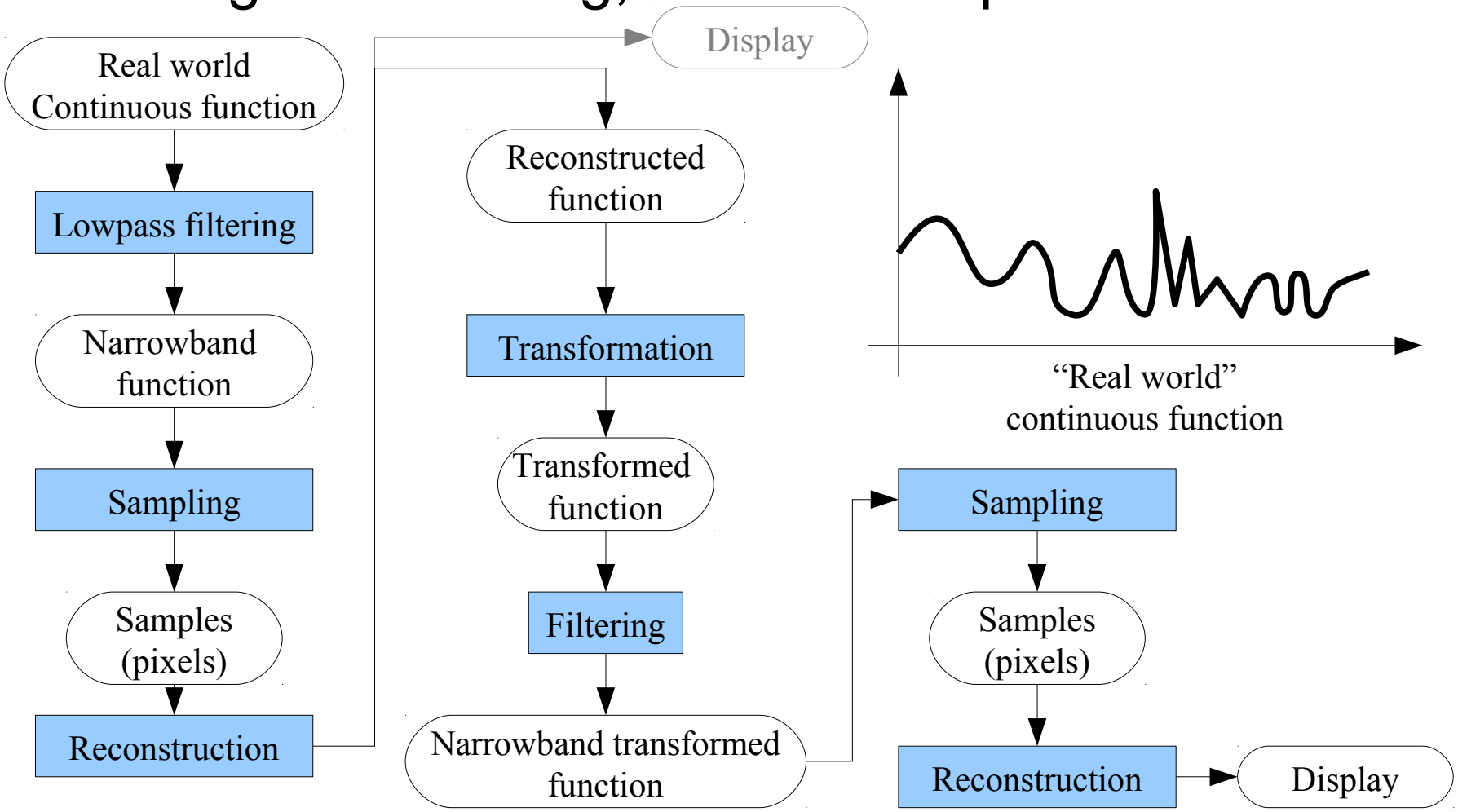
Reminder : 49x49 resolution with 4x4 oversampling

Aliasing

- Common filters used in an image resampling
  - Nearest neighbour
    - Strong aliasing
  - Bilinear interpolation (using the 4 nearest pixels)
    - Softer appearance
  - Bicubic interpolation (using the 16 nearest pixels)
  - Gaussian filter
  - Lanczos
    - Uses an approximation of the sinc() function - which is the perfect low pass filter but suffers from being non-local.

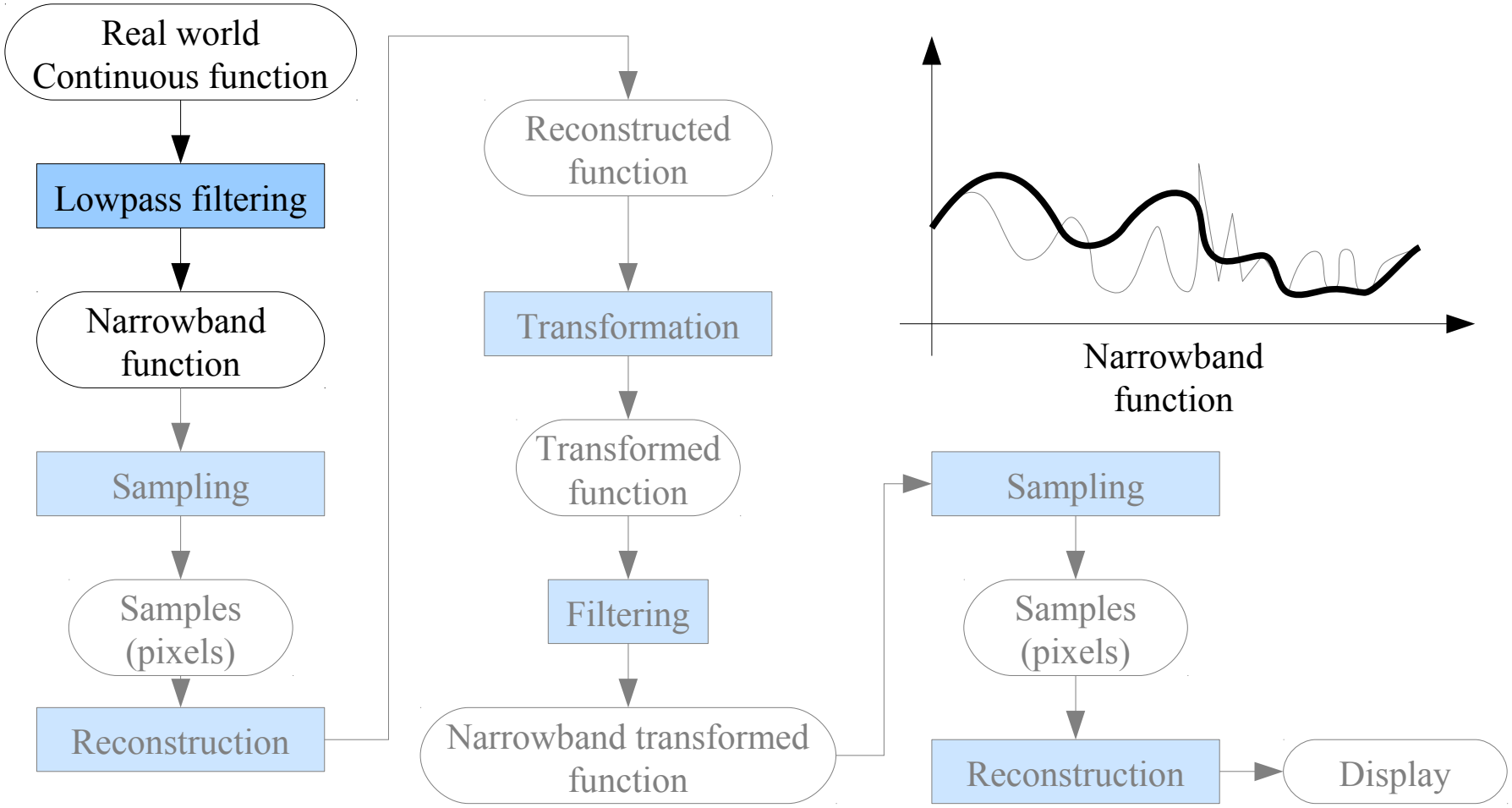
Aliasing

- Image Processing; a series of operations



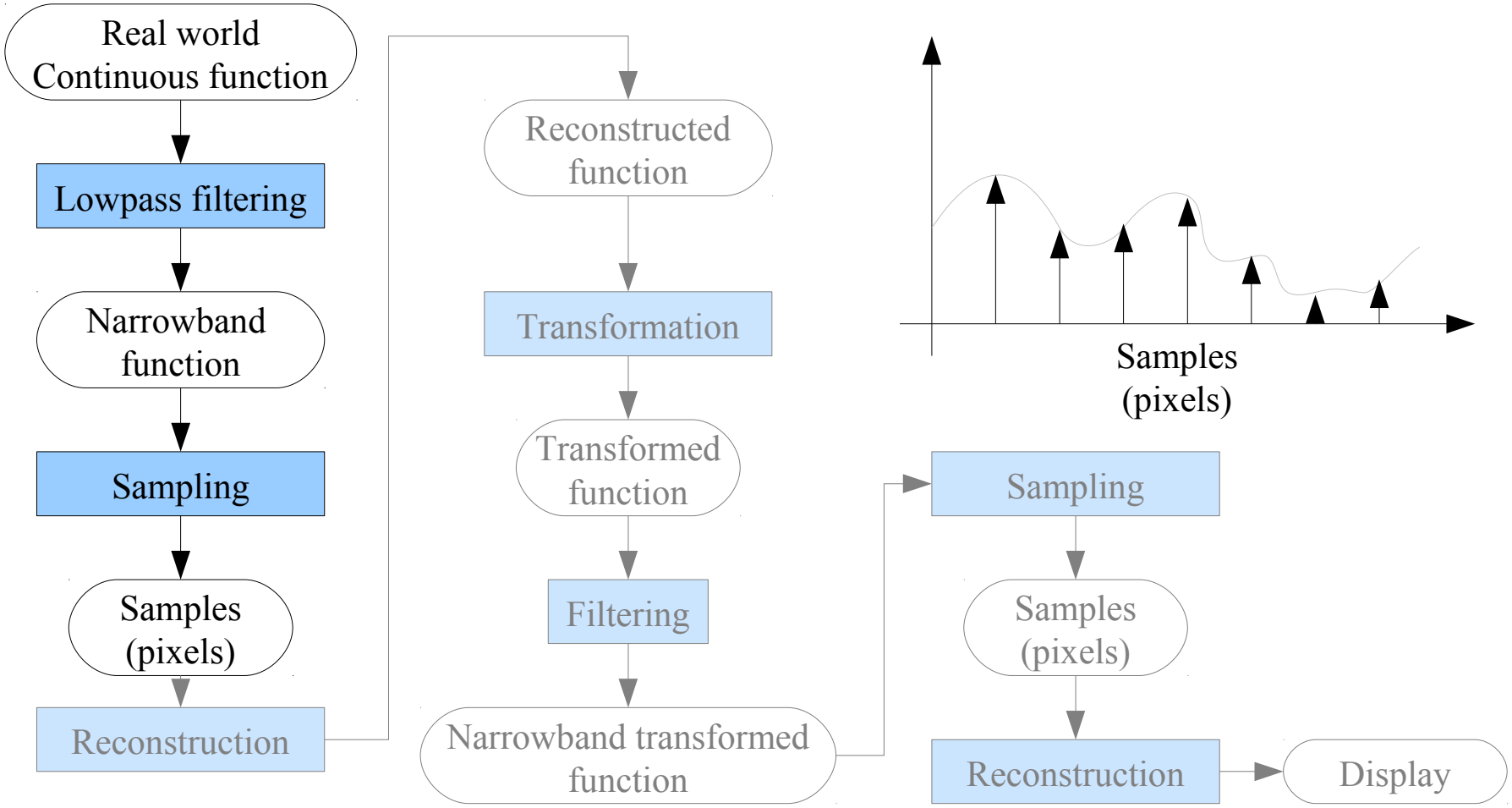
Aliasing

- Image Processing; a series of operations



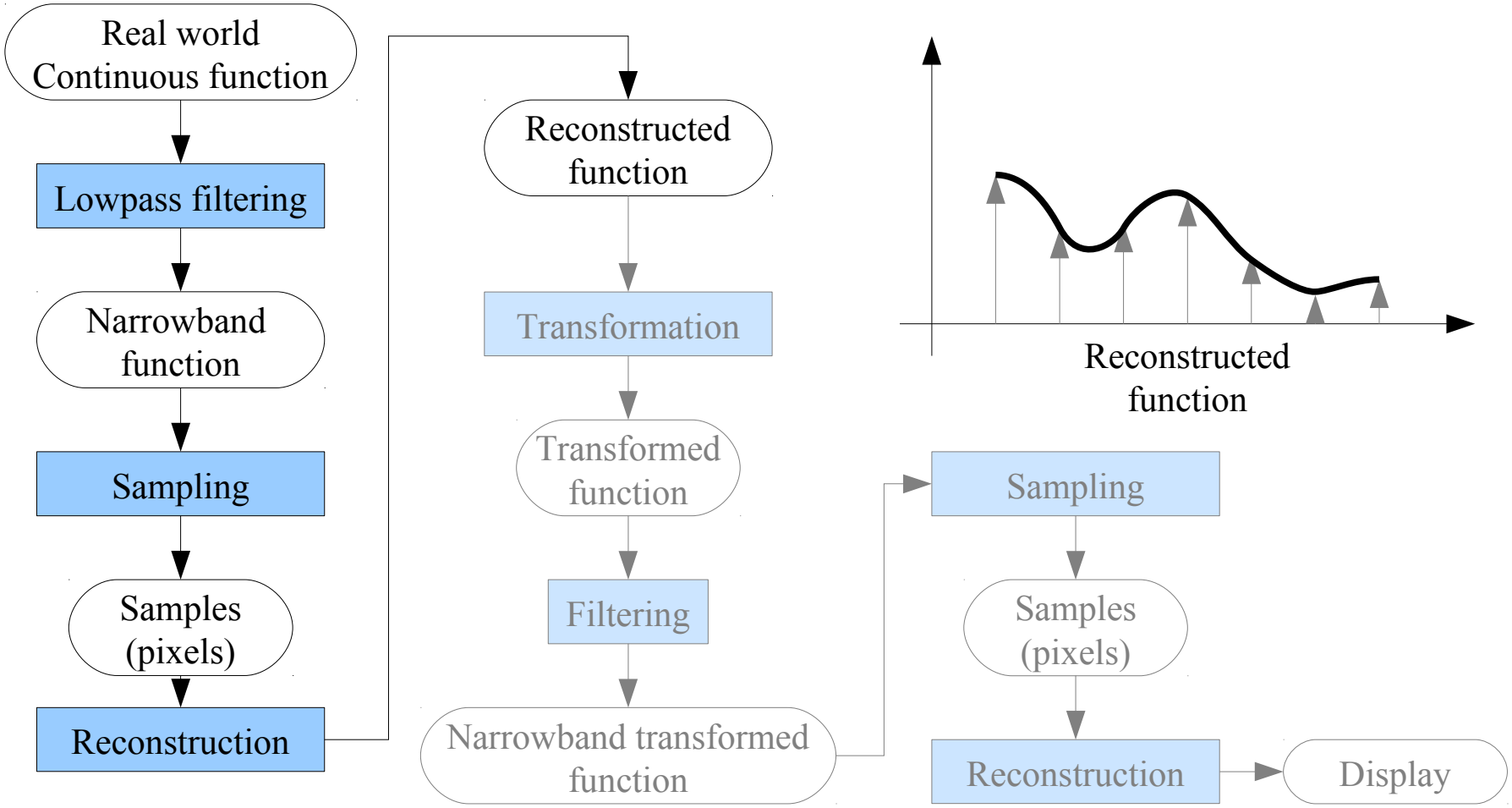
Aliasing

- Image Processing; a series of operations



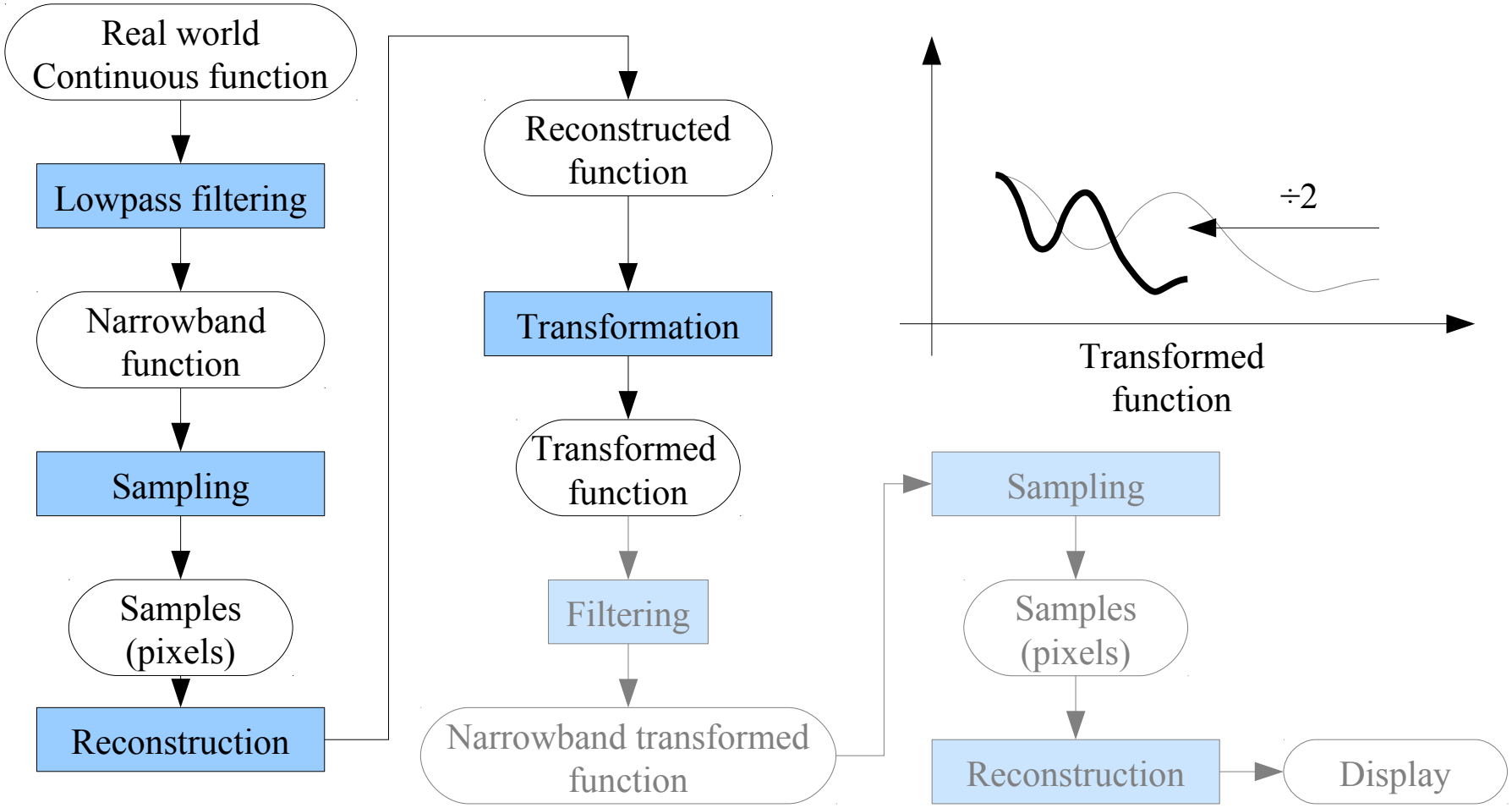
Aliasing

- Image Processing; a series of operations



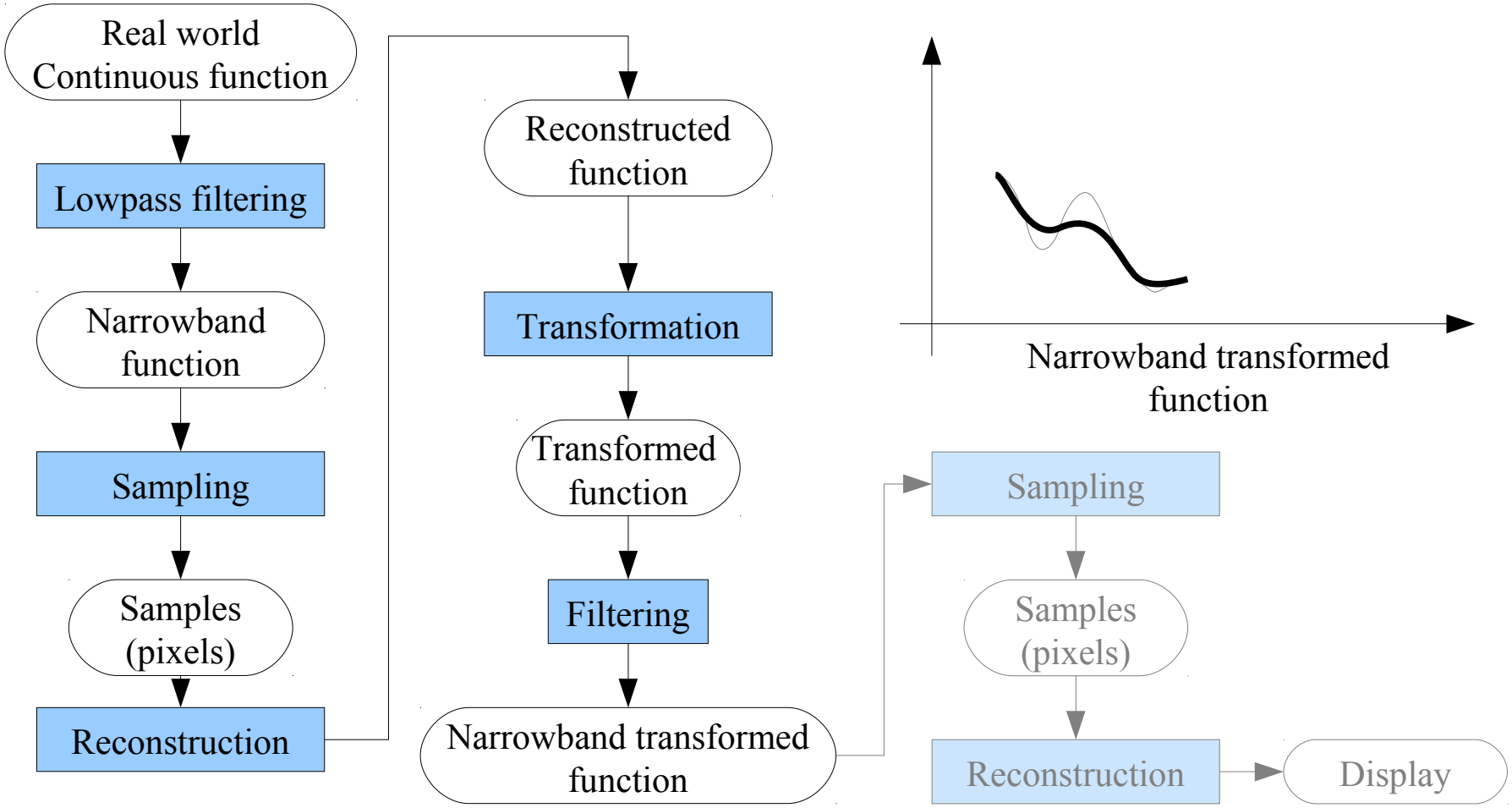
Aliasing

- Image Processing; a series of operations



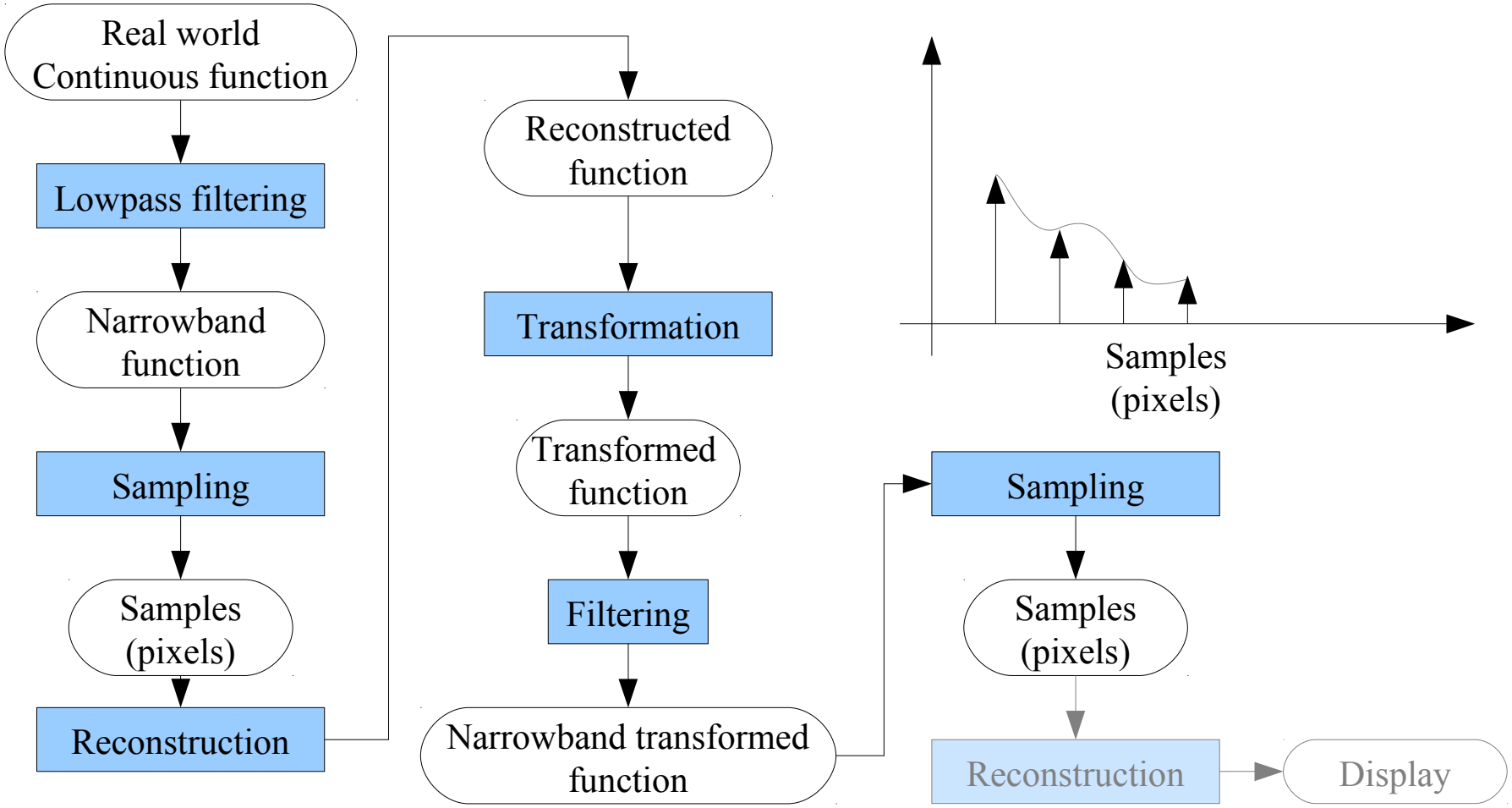
Aliasing

- Image Processing; a series of operations



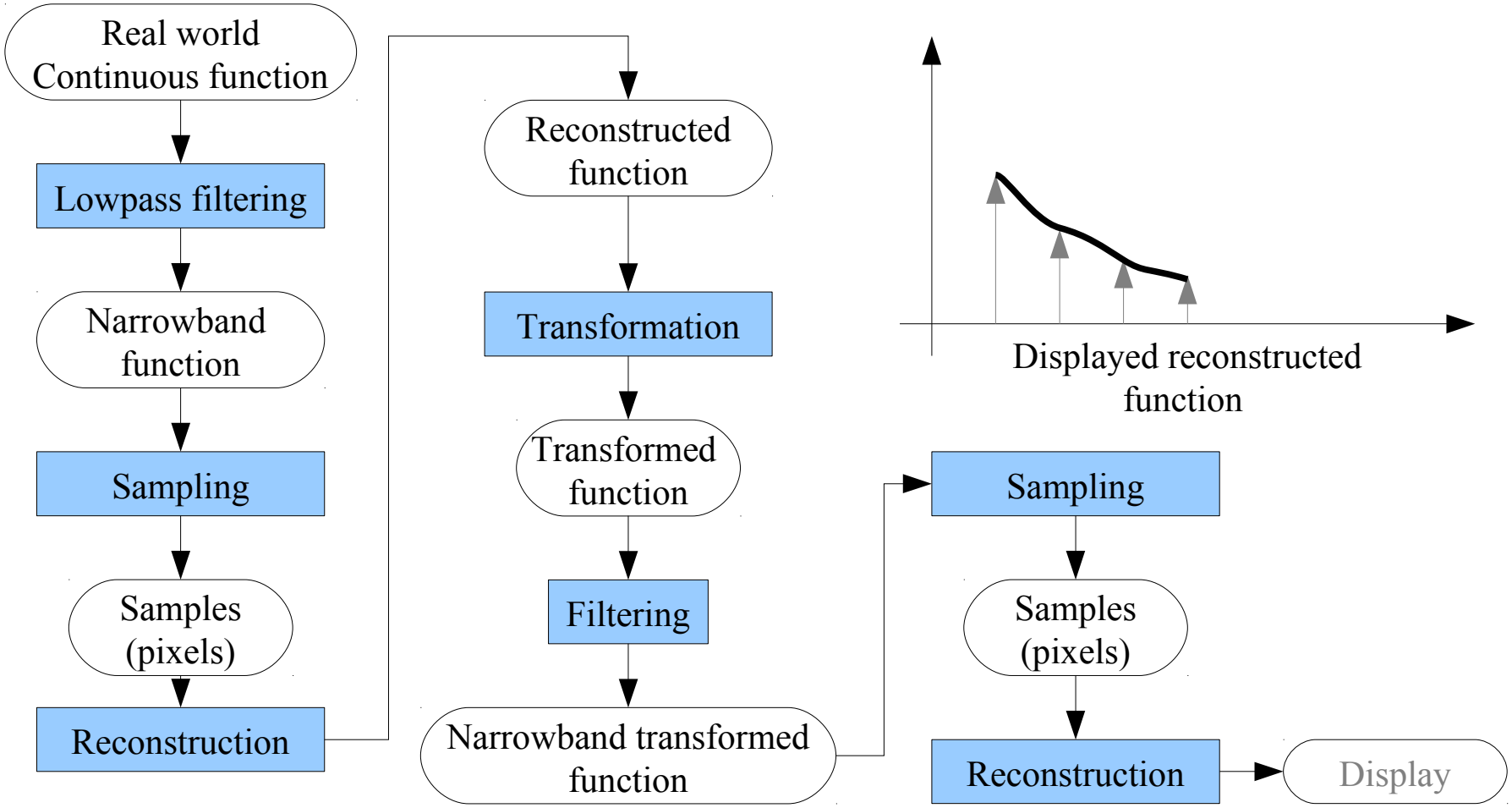
Aliasing

- Image Processing; a series of operations



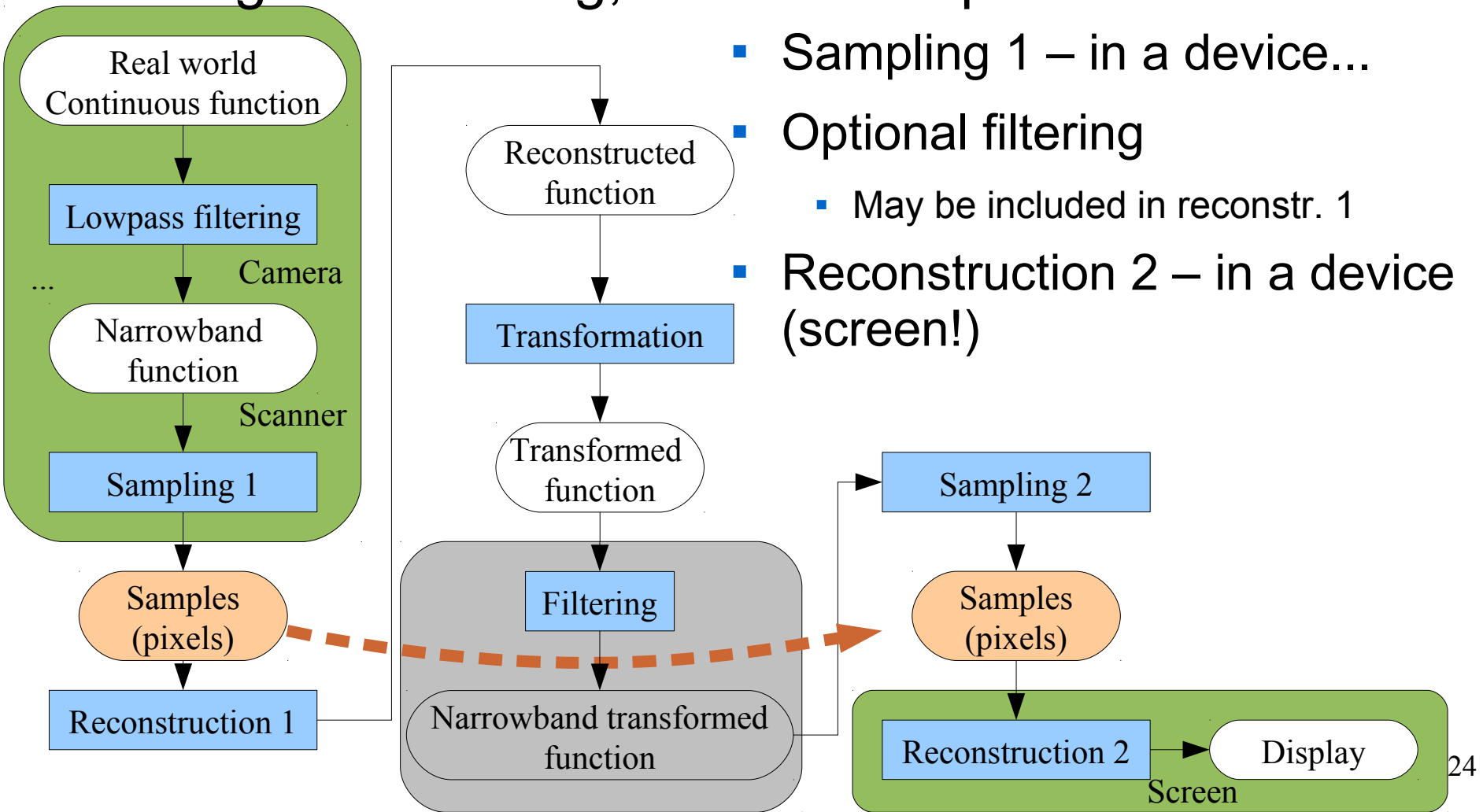
Aliasing

- Image Processing; a series of operations



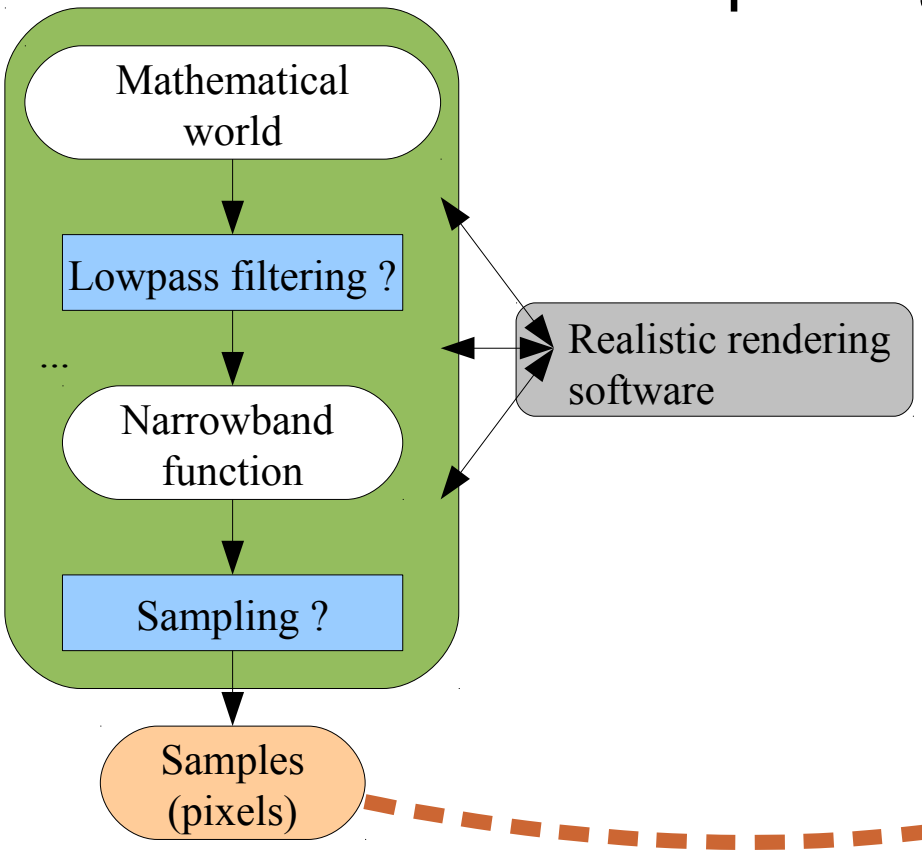
Aliasing

- Image Processing; a series of operations
  - Sampling 1 – in a device...
  - Optional filtering
    - May be included in reconstr. 1
  - Reconstruction 2 – in a device (screen!)



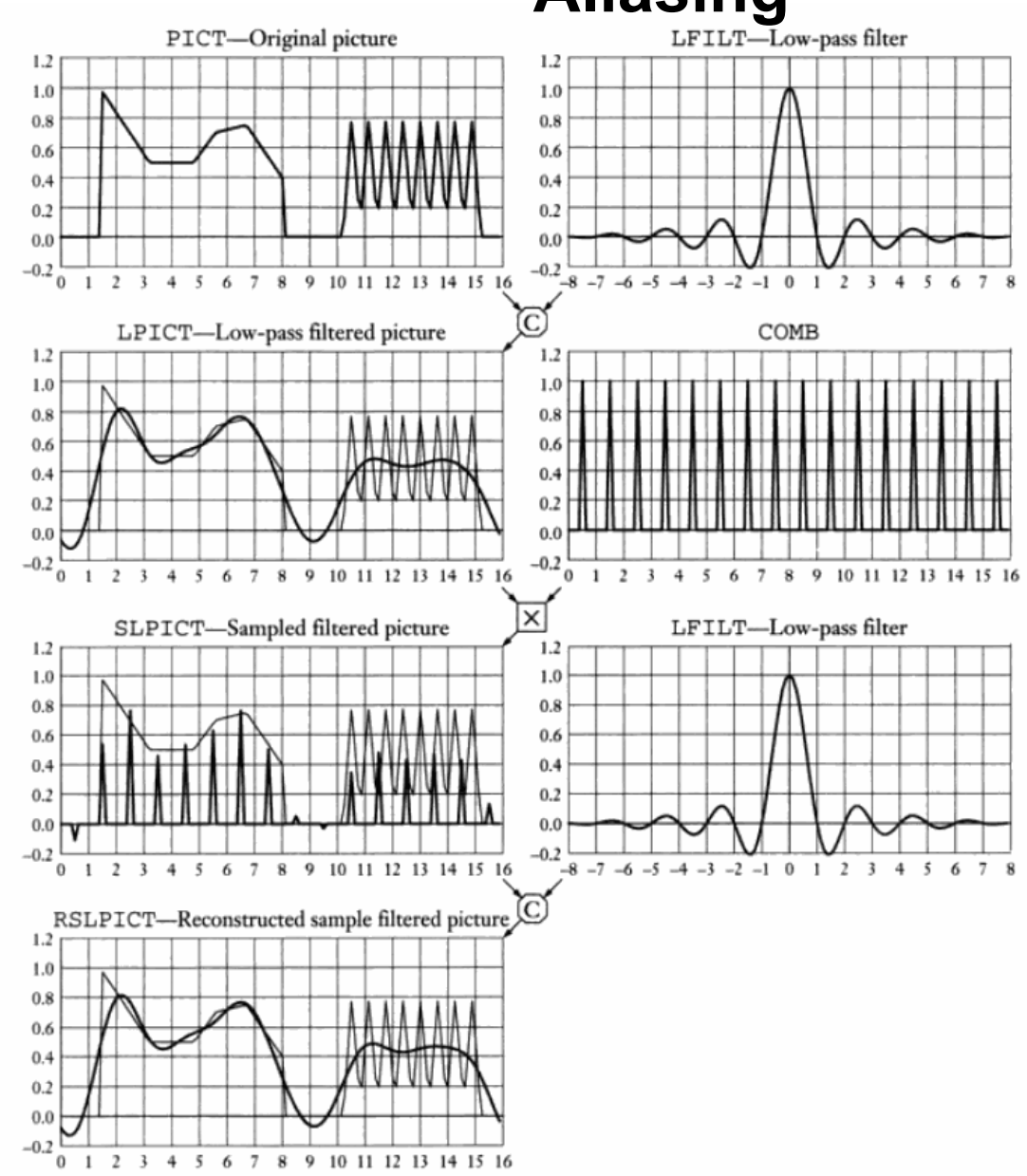
Aliasing

- The case of computer generated images



- Questions arise about low-pass filtering and the sampling of the “mathematical” world that we are representing.
- Difficult to separate these two operations.
- I will discuss about it later.

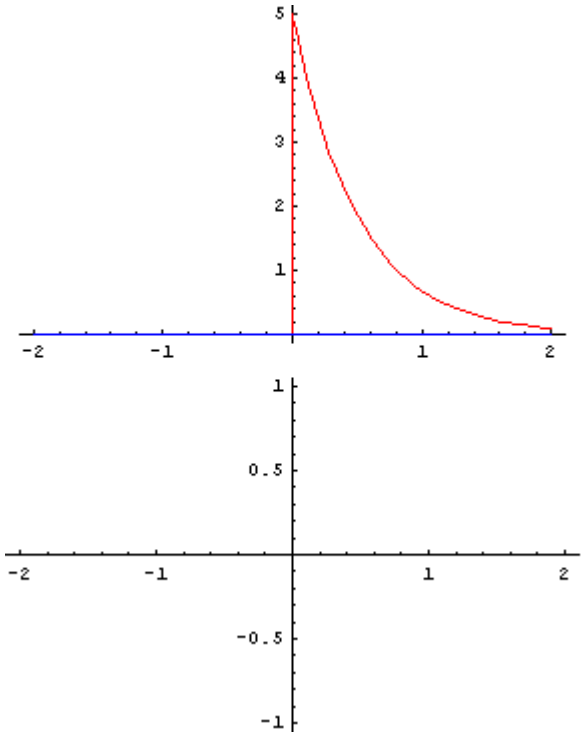
Aliasing


PICT—Original picture
LFILT—Low-pass filter
LPICT—Low-pass filtered picture
COMB
SLPICT—Sampled filtered picture
LFILT—Low-pass filter
RSLPICT—Reconstructed sample filtered picture
C
×
C

Aliasing

- Convolution

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(t-\tau)\, g(\tau)\, d\tau$$

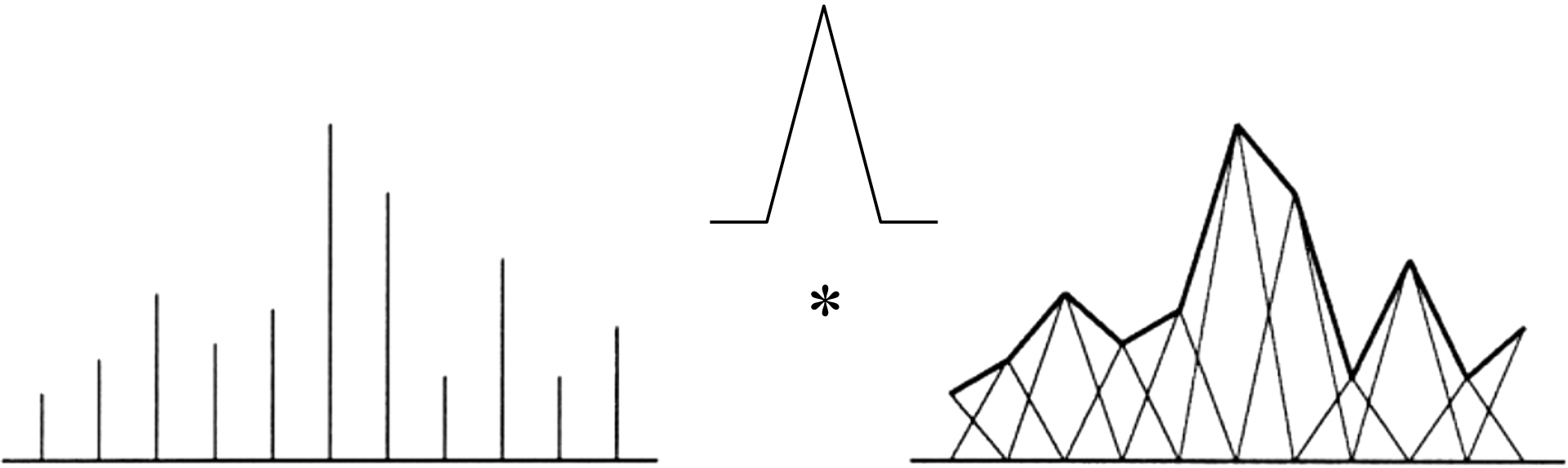


Aliasing

- Reconstruction
  - From samples
  - Convolution with a certain function
    - Linear, bicubic, Gaussian, etc...
    - Interpolates the signal where it no longer exists (between samples)
    - Back to a continuous signal ...

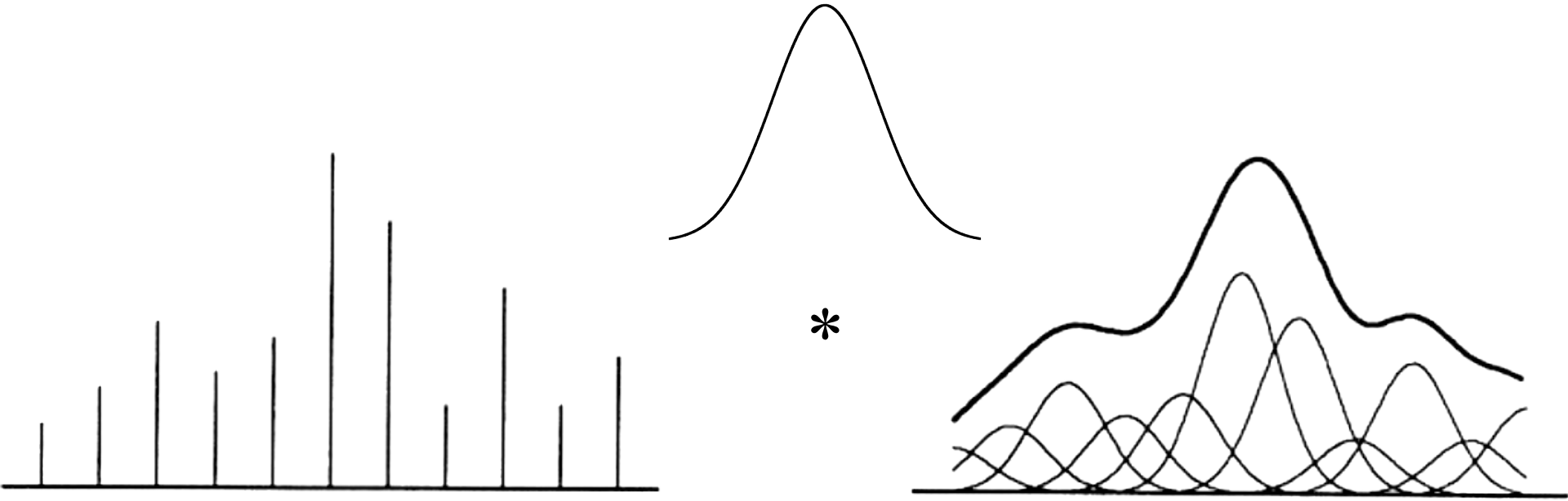
Aliasing

- Reconstruction with a linear filter (« hat »)
  - Each sample is "multiplied" by the hat function and the sum constitutes the reconstructed function



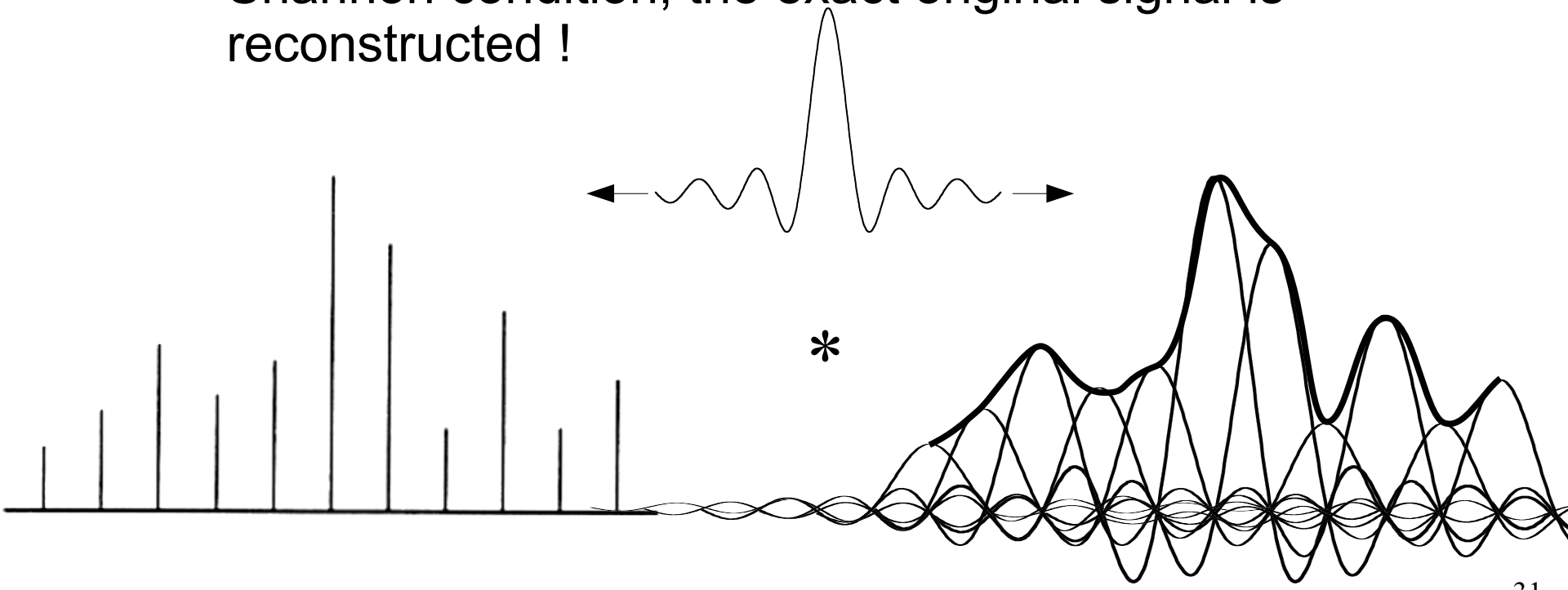
Aliasing

- Reconstruction with a Gaussian filter
  - Each sample is "multiplied" by the Gaussian function and the sum constitutes the reconstructed function



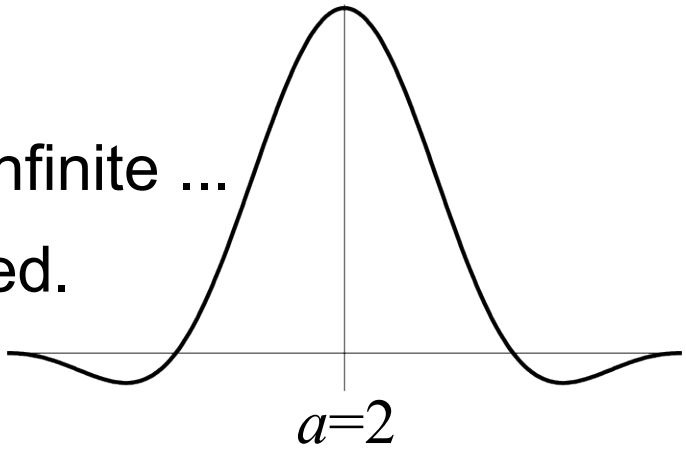
Aliasing

- Reconstruction with sinc $sinc(x) = \frac{\sin \pi x}{\pi x}$
  - Convolution with a cardinal sine that has an infinite support : provided that the original signal meets the Shannon condition, the exact original signal is reconstructed !



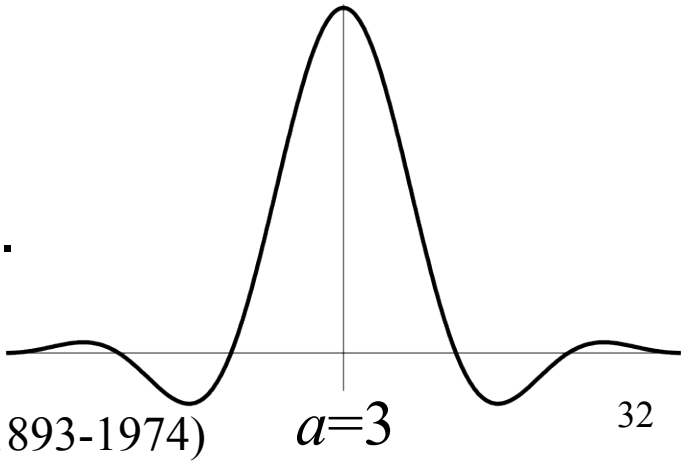
Aliasing

- The sinc has an infinite support
  - In theory, the computing effort is infinite ...
  - In practice, the function is truncated.



$$L(x)=\begin{cases} sinc(x)\,sinc(x/a) & \text{if} \quad -a<x<a\,,\ x\neq 0 \\ 1 & \text{if} \quad x=0 \\ 0 & \text{otherwise} \end{cases}$$

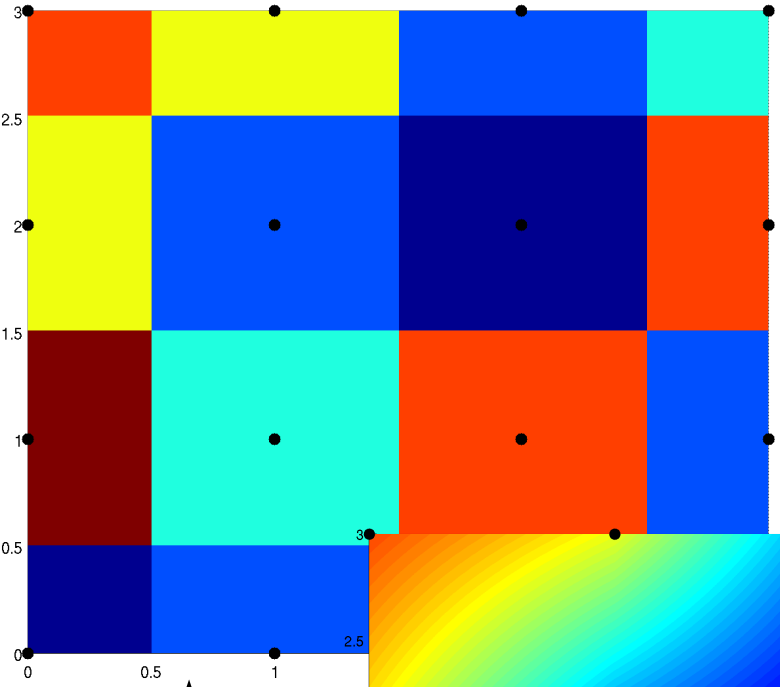
- This is the famous “Lanczos” filter.



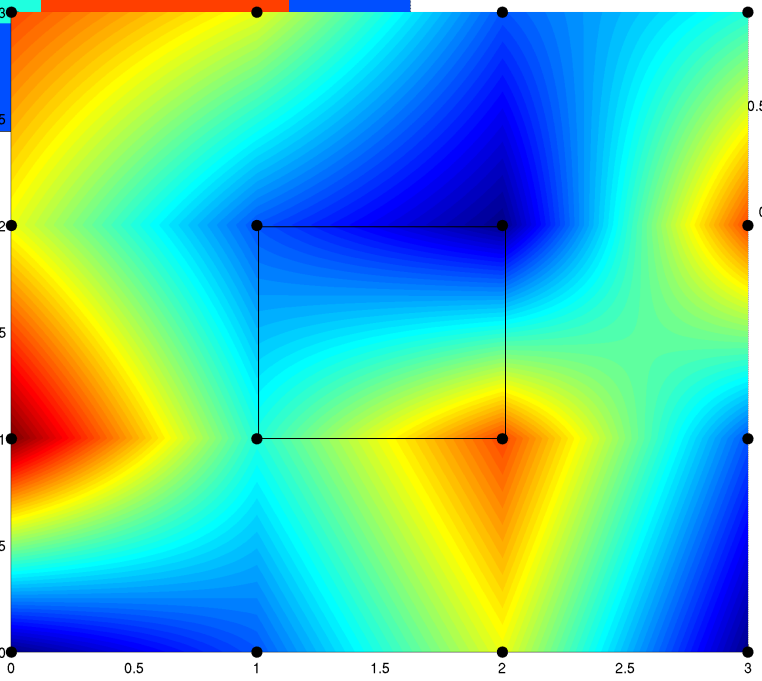
*Cornelius Lanczos, Hungarian mathematician (1893-1974)

Computer Graphics

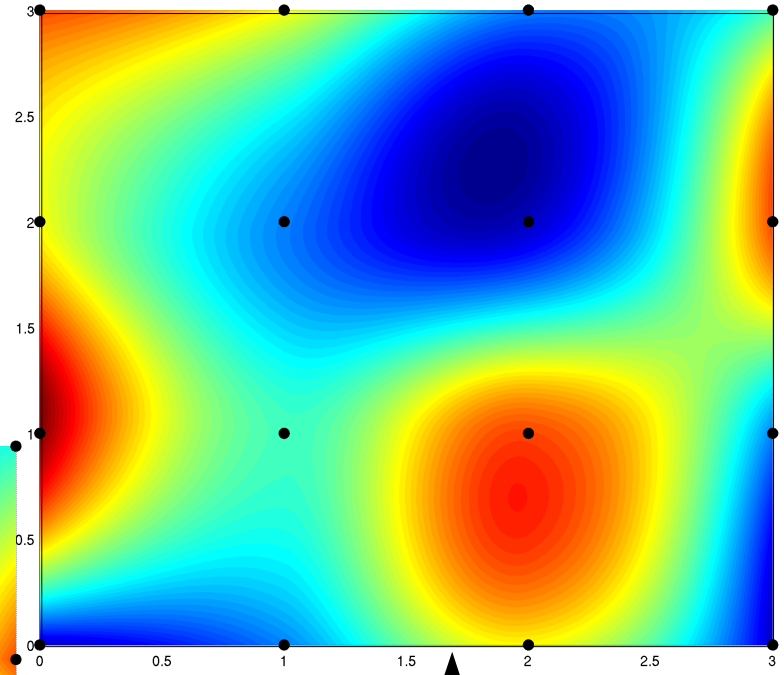
Aliasing



Nearest neighbor interpolation



Bilinear interpolation



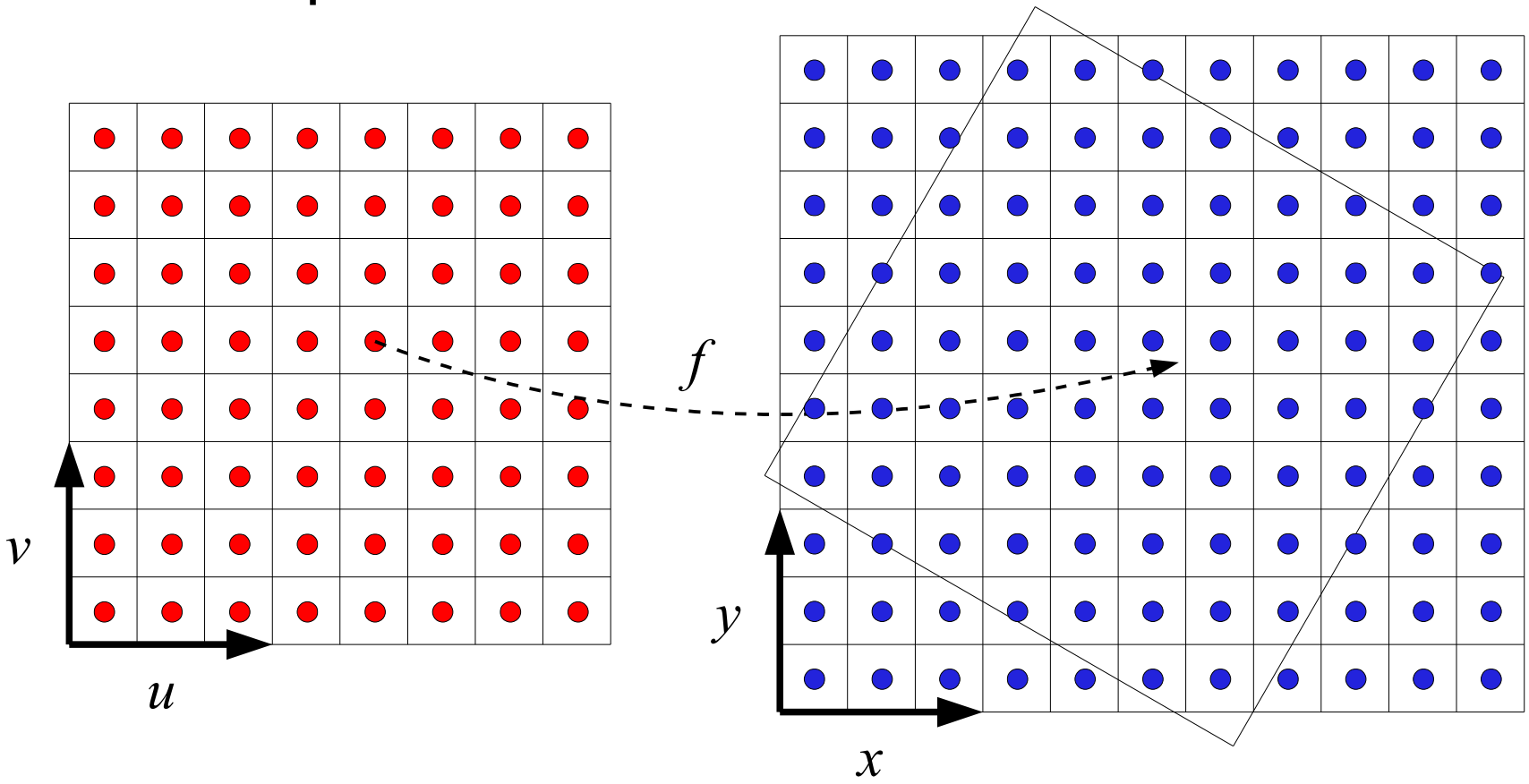
Bicubic interpolation

Aliasing

- Transformation
  - Change the position of the image / or the samples
    - Rotation
    - Scaling
    - Etc...
  - In cases where the targeted sample density is lower, filtering with a low-pass filter is needed before resampling
  - In cases where the targeted sample density is identical or finer, a simple resampling after reconstruction is sufficient

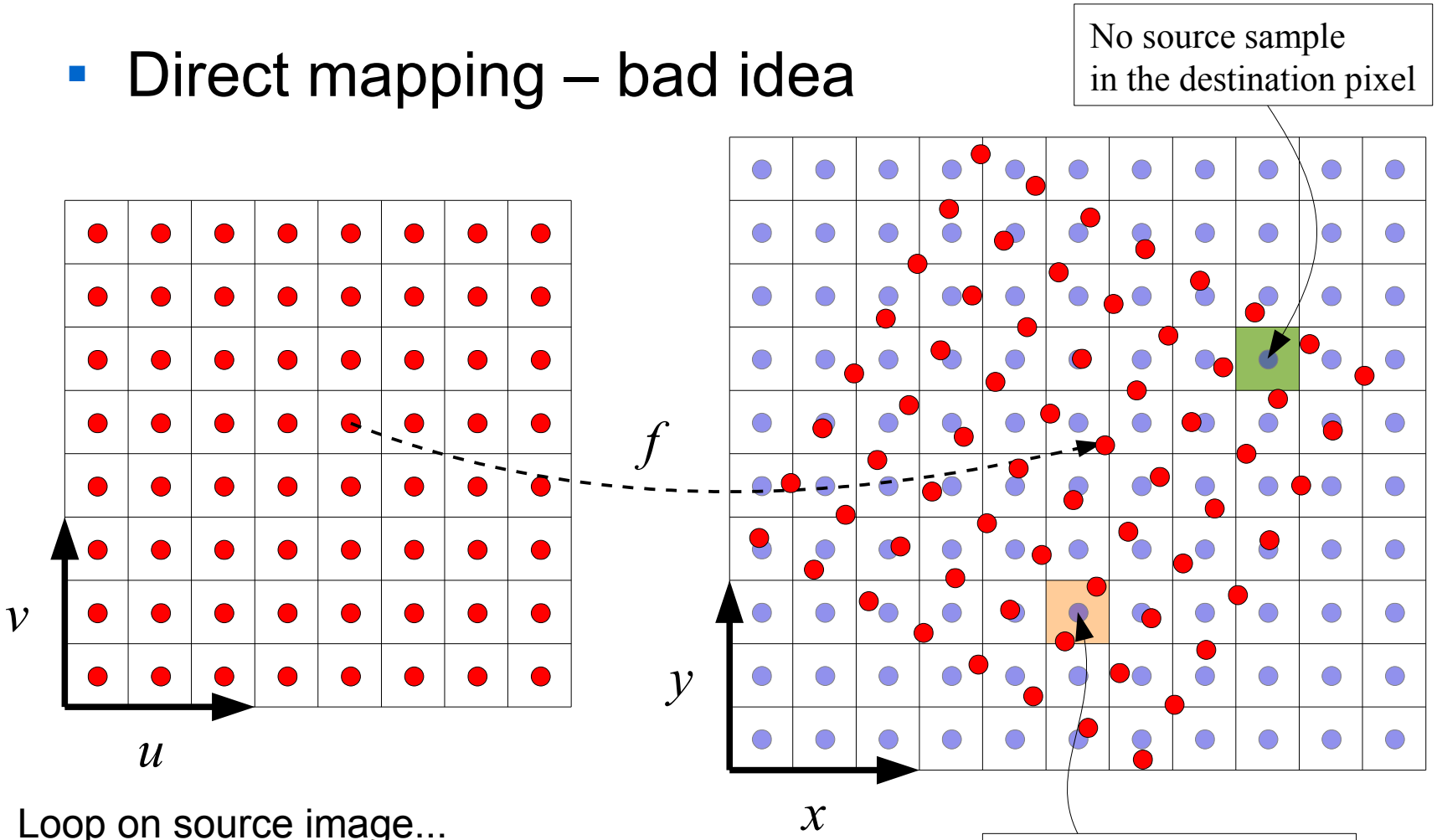
Aliasing

- Example with a rotation



Aliasing

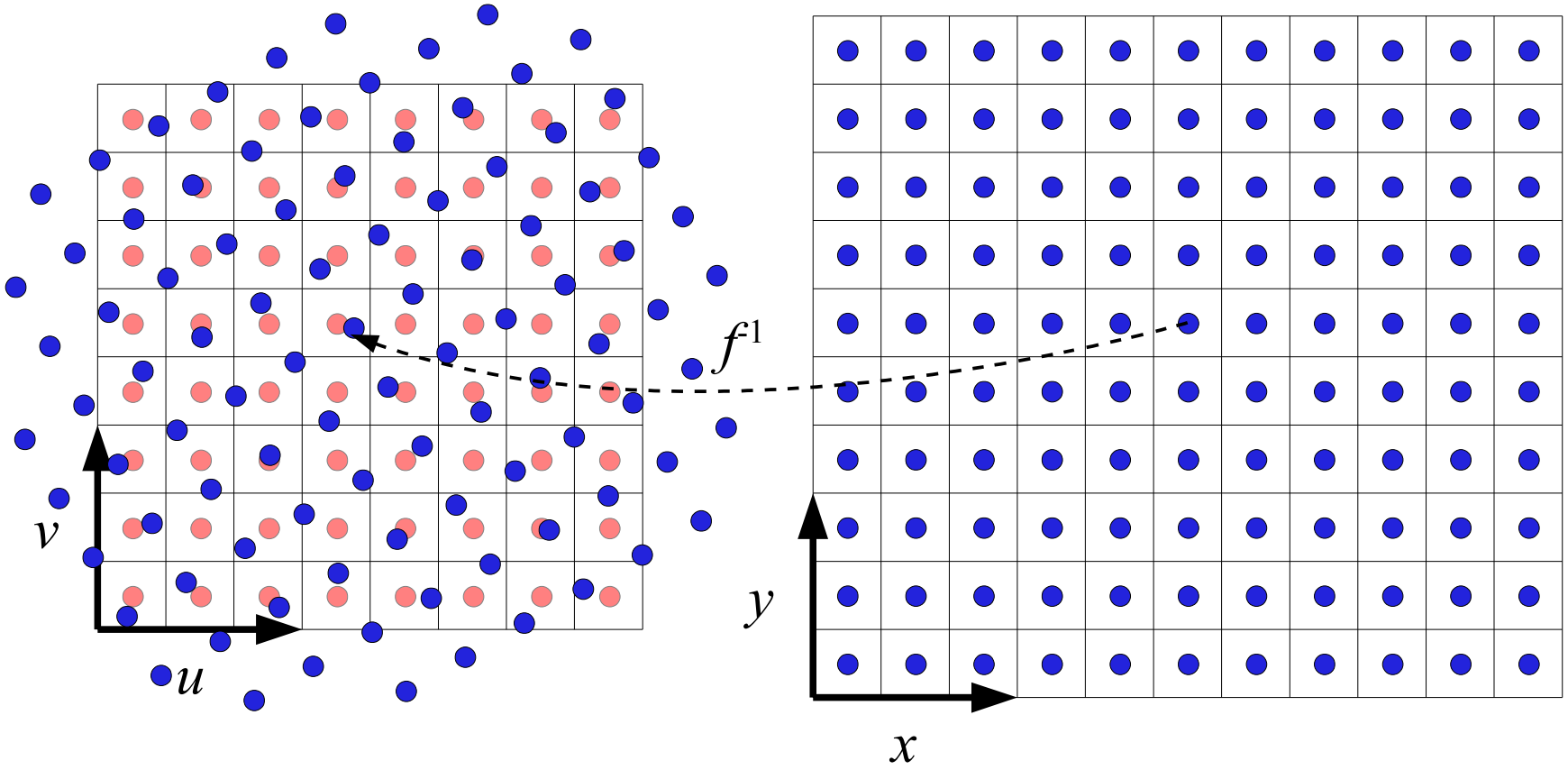
- Direct mapping – bad idea



Loop on source image...
but difficult to “reconstruct” the target image (it is an inverse problem)

Aliasing

- Inverse mapping



Loop on pixels of the target image ...

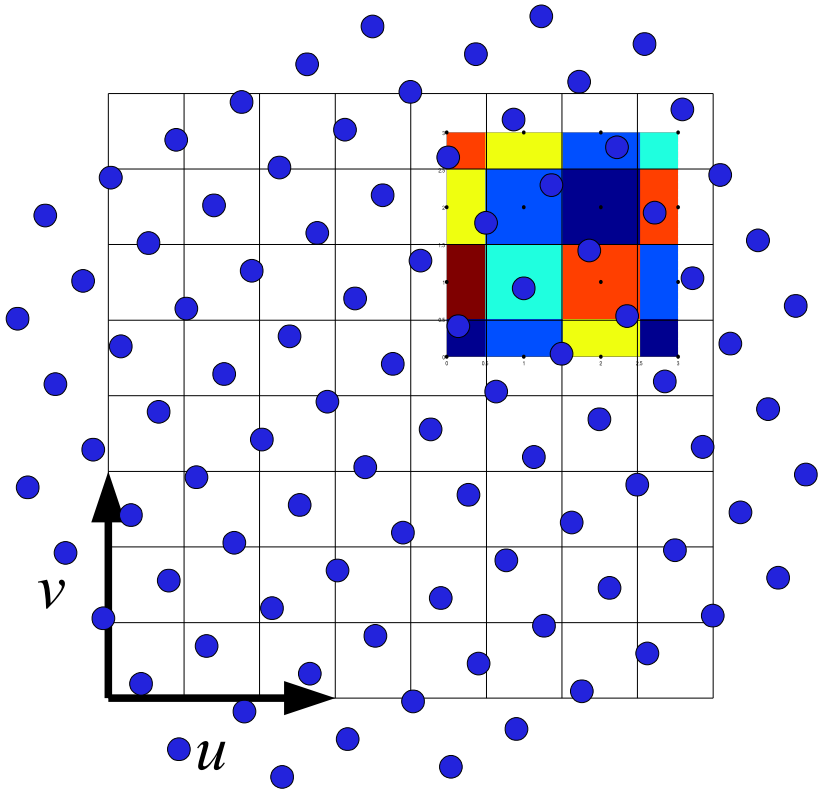
- Resampling is unavoidable
- A reconstruction of the source image is used here

Aliasing

- The reconstruction is crucial to the quality of the target image
  - Nearest neighbour – lots of aliasing
  - Bilinear – not much aliasing but blurred image (loss of details)
  - Bicubic and « Lanczos » - better (but more computationally expensive)

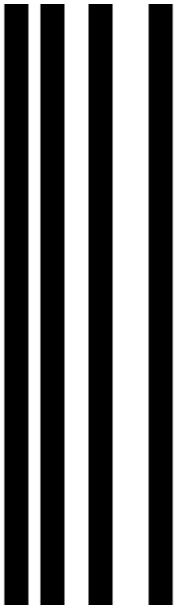
Aliasing

- Nearest neighboor :



Aliasing

- Serie of 36 rotations of 5° → 180° followed by a mirroring (without any loss)
  - Original images magnified 10x

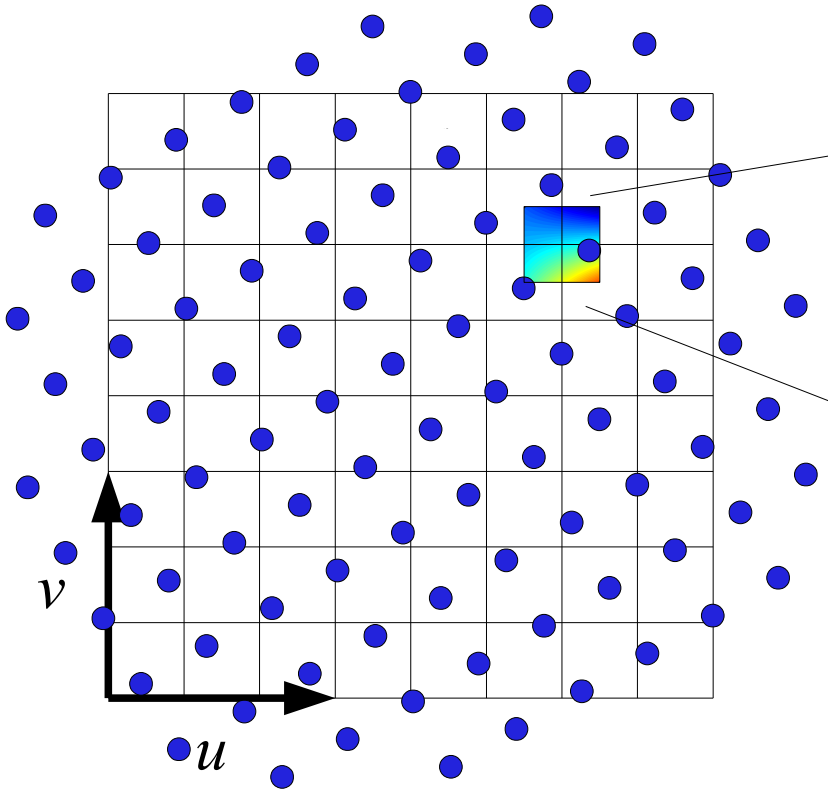


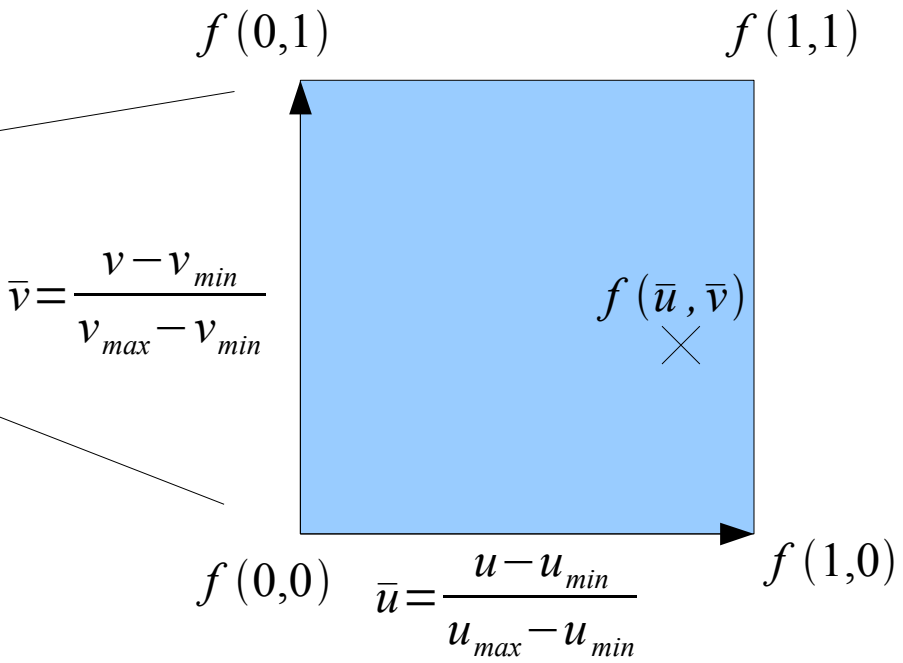
Aliasing

- « Nearest neighbour » filter

Aliasing

- Bilinear :





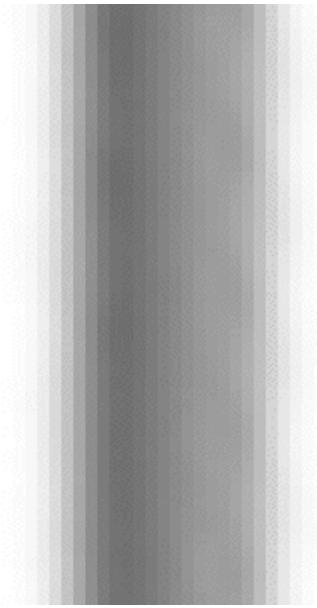
$$\bar{v} = \frac{v - v_{min}}{v_{max} - v_{min}}$$

$$\bar{u} = \frac{u - u_{min}}{u_{max} - u_{min}}$$

$$f(\bar{u},\bar{v}) = f(0,0)(1-\bar{u})(1-\bar{v}) + f(1,0)\bar{u}(1-\bar{v}) + f(0,1)(1-\bar{u})\bar{v} + f(1,1)\bar{u}\bar{v}$$

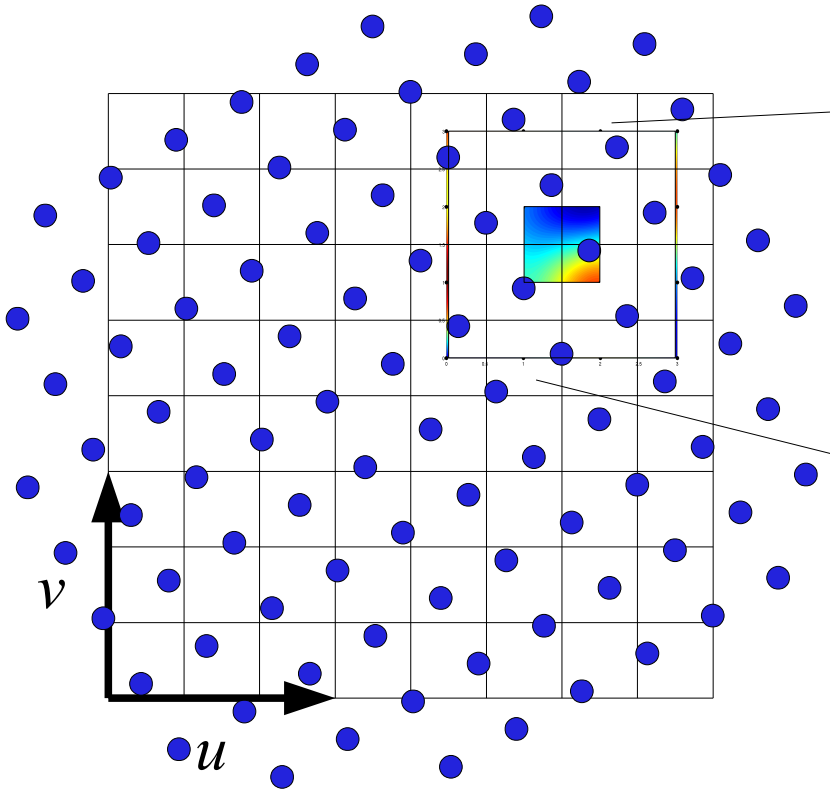
Aliasing

- « Bilinear » filter

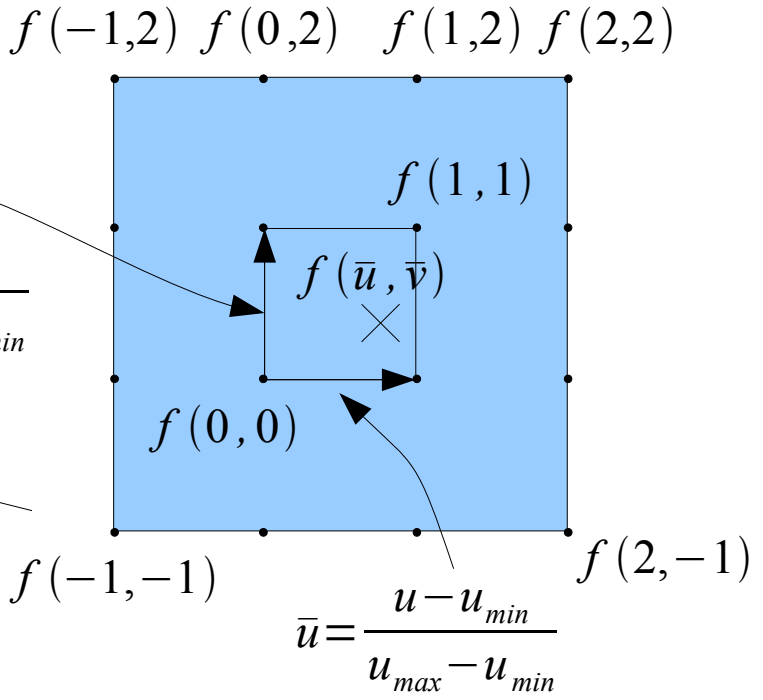


Aliasing

- Bicubic :



$$\bar{v} = \frac{v - v_{min}}{v_{max} - v_{min}}$$



$$\bar{u} = \frac{u - u_{min}}{u_{max} - u_{min}}$$

$$\begin{aligned} f(\bar{u},\bar{v}) = {} & a_{00} + a_{01}\bar{v} + a_{02}\bar{v}^2 + a_{03}\bar{v}^3 \\ & + a_{10}\bar{u} + a_{11}\bar{u}\bar{v} + a_{12}\bar{u}\bar{v}^2 + a_{13}\bar{u}\bar{v}^3 \\ & + a_{20}\bar{u}^2 + a_{21}\bar{u}^2\bar{v} + a_{22}\bar{u}^2\bar{v}^2 + a_{23}\bar{u}^2\bar{v}^3 \\ & + a_{30}\bar{u}^3 + a_{31}\bar{u}^3\bar{v} + a_{32}\bar{u}^3\bar{v}^2 + a_{33}\bar{u}^3\bar{v}^3 \end{aligned}$$

Aliasing

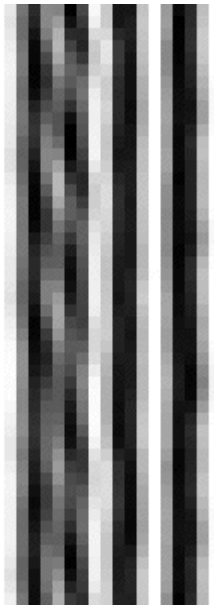
Here , $p[i][j]=f(i\text{-}1, j\text{-}1)$

```
double bicubicInterpolate (double[][] p, double u, double v)
{
double a00 = p[1][1];
double a01 = p[1][2] - p[1][1]/2 - p[1][0]/3 - p[1][3]/6;
double a02 = p[1][0]/2 - p[1][1] + p[1][2]/2;
double a03 = p[1][1]/2 - p[1][0]/6 - p[1][2]/2 + p[1][3]/6;
double a10 = p[2][1] - p[1][1]/2 - p[0][1]/3 - p[3][1]/6;
double a11 = p[0][0]/9 + p[0][1]/6 - p[0][2]/3 + p[0][3]/18 + p[1][0]/6 + p[1][1]/4 - p[1][2]/2 + p[1][3]/12 -
p[2][0]/3 - p[2][1]/2 + p[2][2] - p[2][3]/6 + p[3][0]/18 + p[3][1]/12 - p[3][2]/6 + p[3][3]/36;
double a12 = p[0][1]/3 - p[0][0]/6 - p[0][2]/6 - p[1][0]/4 + p[1][1]/2 - p[1][2]/4 + p[2][0]/2 - p[2][1] + p[2]
[2]/2 - p[3][0]/12 + p[3][1]/6 - p[3][2]/12;
double a13 = p[0][0]/18 - p[0][1]/6 + p[0][2]/6 - p[0][3]/18 + p[1][0]/12 - p[1][1]/4 + p[1][2]/4 - p[1][3]/12 -
p[2][0]/6 + p[2][1]/2 - p[2][2]/2 + p[2][3]/6 + p[3][0]/36 - p[3][1]/12 + p[3][2]/12 - p[3][3]/36;
double a20 = p[0][1]/2 - p[1][1] + p[2][1]/2;
double a21 = p[0][2]/2 - p[0][1]/4 - p[0][0]/6 - p[0][3]/12 + p[1][0]/3 + p[1][1]/2 - p[1][2] + p[1][3]/6 - p[2]
[0]/6 - p[2][1]/4 + p[2][2]/2 - p[2][3]/12;
double a22 = p[0][0]/4 - p[0][1]/2 + p[0][2]/4 - p[1][0]/2 + p[1][1] - p[1][2]/2 + p[2][0]/4 - p[2][1]/2 + p[2]
[2]/4;
double a23 = p[0][1]/4 - p[0][0]/12 - p[0][2]/4 + p[0][3]/12 + p[1][0]/6 - p[1][1]/2 + p[1][2]/2 - p[1][3]/6 -
p[2][0]/12 + p[2][1]/4 - p[2][2]/4 + p[2][3]/12;
double a30 = p[1][1]/2 - p[0][1]/6 - p[2][1]/2 + p[3][1]/6;
double a31 = p[0][0]/18 + p[0][1]/12 - p[0][2]/6 + p[0][3]/36 - p[1][0]/6 - p[1][1]/4 + p[1][2]/2 - p[1][3]/12 +
p[2][0]/6 + p[2][1]/4 - p[2][2]/2 + p[2][3]/12 - p[3][0]/18 - p[3][1]/12 + p[3][2]/6 - p[3][3]/36;
double a32 = p[0][1]/6 - p[0][0]/12 - p[0][2]/12 + p[1][0]/4 - p[1][1]/2 + p[1][2]/4 - p[2][0]/4 + p[2][1]/2 -
p[2][2]/4 + p[3][0]/12 - p[3][1]/6 + p[3][2]/12;
double a33 = p[0][0]/36 - p[0][1]/12 + p[0][2]/12 - p[0][3]/36 - p[1][0]/12 + p[1][1]/4 - p[1][2]/4 + p[1][3]/12
+ p[2][0]/12 - p[2][1]/4 + p[2][2]/4 - p[2][3]/12 - p[3][0]/36 + p[3][1]/12 - p[3][2]/12 + p[3][3]/36;

double u2 = u * u; double u3 = u2 * u; double v2 = v * v; double v3 = v2 * v;
return a00      + a01 * v      + a02 * v2      + a03 * v3 +
       a10 * u  + a11 * u * v  + a12 * u * v2  + a13 * u * v3 +
       a20 * u2 + a21 * u2 * v + a22 * u2 * v2 + a23 * u2 * v3 +
       a30 * u3 + a31 * u3 * v + a32 * u3 * v2 + a33 * u3 * v3;
}
```

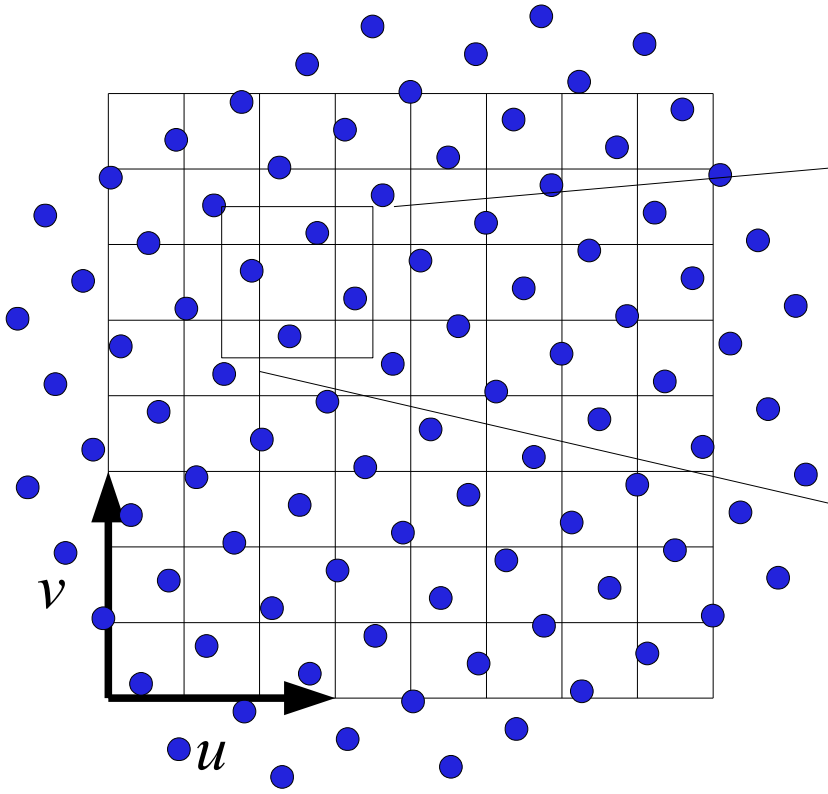
Aliasing

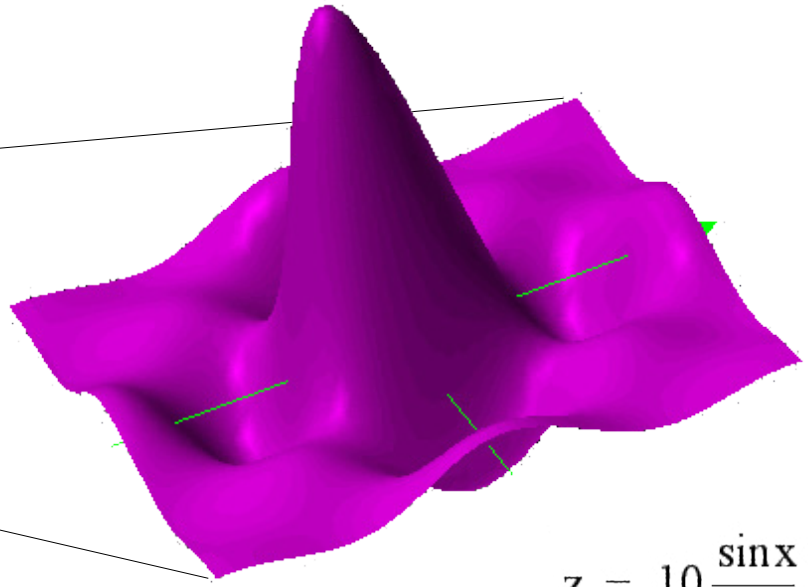
- « Bicubic » filter



Aliasing

- Lanczos :

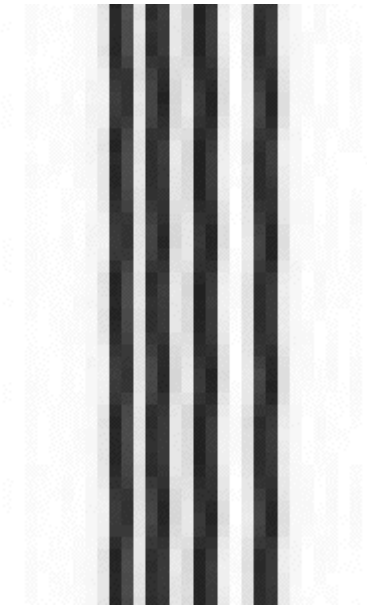




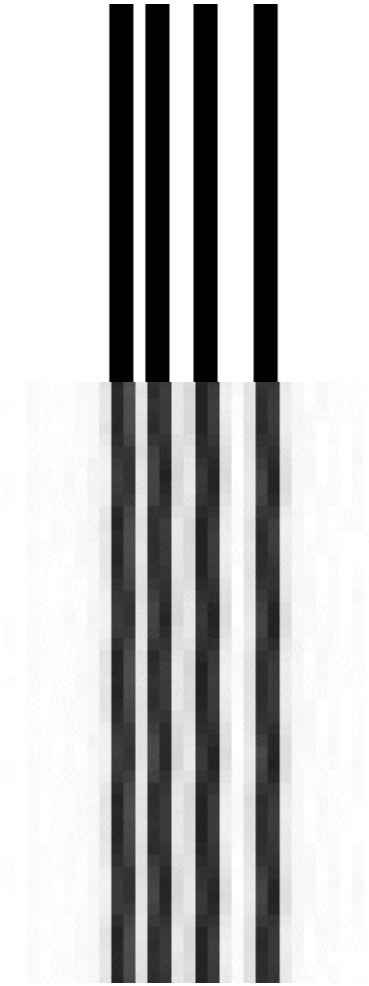
$$z = 10 \frac{\sin x}{x} \frac{\sin y}{y}$$

Aliasing

- « Lanczos » filter

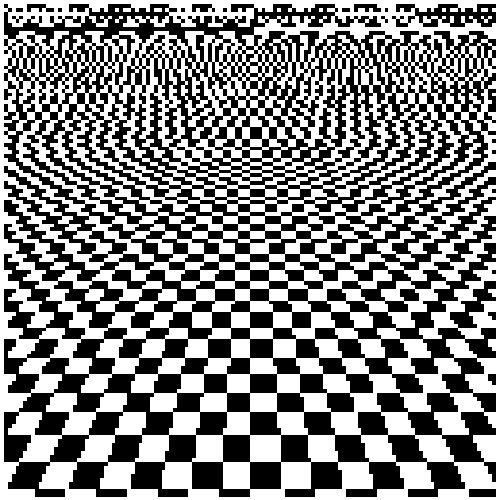


Aliasing

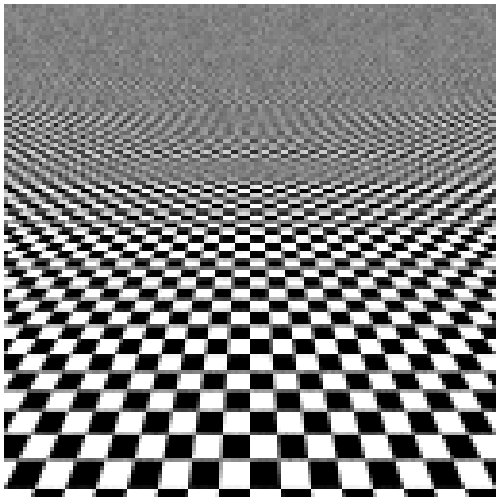


Aliasing

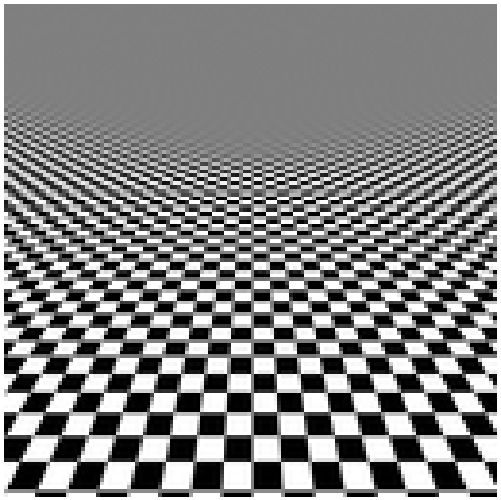
- Another example



Without antialiasing
nearest neighbour



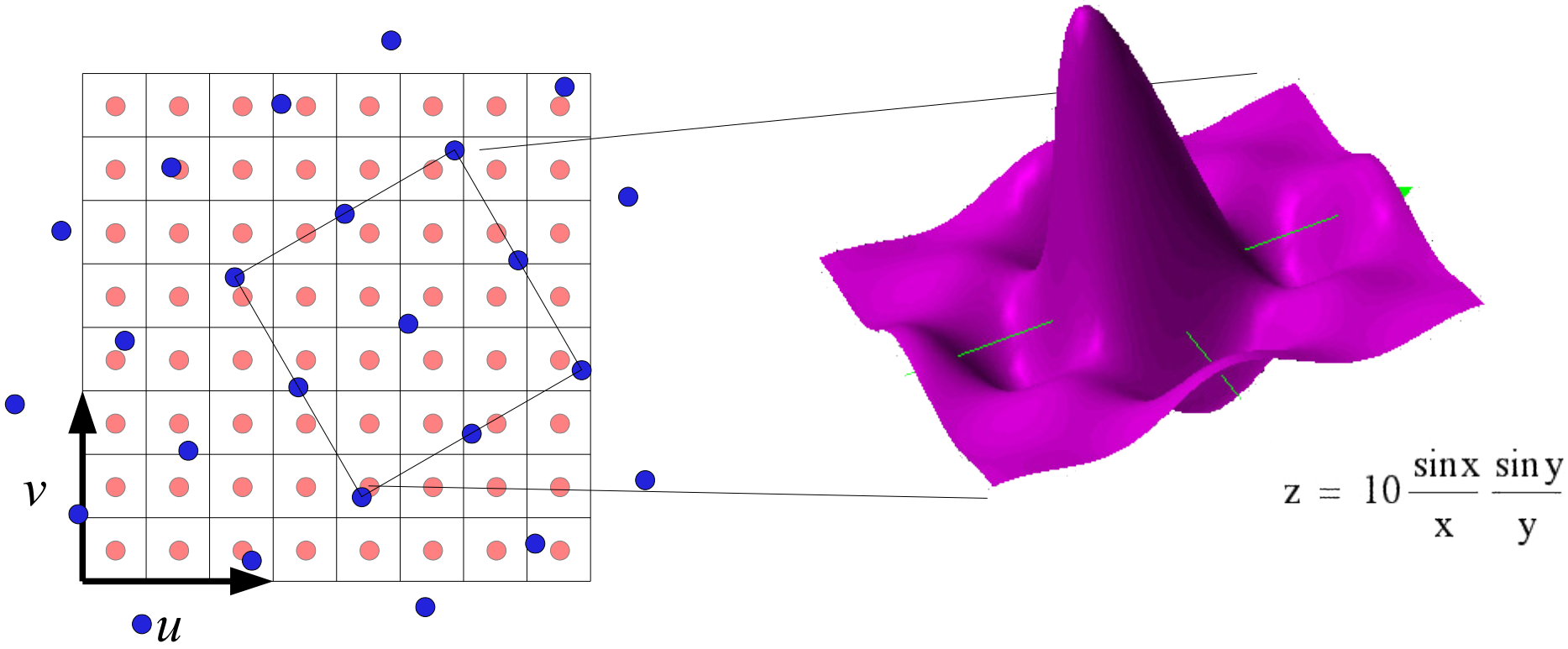
Antialiasing
by oversampling 4
and simple average



Antialiasing by
Lanczos filtering
(ideal but costly)

Aliasing

- Case of downsampling



The input data must be filtered with a low pass filter, that is matched with the resolution of the destination image

Image storage

Image storage

- Idea : images are not random
  - We can take advantage of the structure to store images
  - Two approaches
    - Vectorised images
    - Discretized images
      - Compression without losses
      - Compression with controled losses

tiff gif png

jpg svg cin

exr

bmp

Image storage

- TIFF : universal format but sometimes partially implemented
- JPG : limited to 8bits/channel, DCT compression with losses (WWW)
- PNG : open format, 1/2/4/8 indexed bits, 8/16 bits/channel ; alpha channel (transparency), LZW type compression – no patent (WWW)
- GIF : indexed 8 bits, transparency (1 bit), LZW type compression, possible animation, expired patent (WWW)
- SVG : vector images
- CIN : old format « cineon » 10 bits / channel, used for special effects. Lossless compression
- EXR : open format Lucasfilm (ILM): 16/32 bits by channel in floating point, lossless compression
- BMP : old Windows format without compression limited to 8 bits by channel + transparency

Image storage

- How to choose ?
  - Outline drawing, to be scaled --- vector format
  - Images in general, sampled (bitmap) format

Image storage

- Vector images
  - Generally no “geometric” compression
  - A typical example : character fonts destined to be enlarged
  - Ideal as a format for line drawings

WMF : windows metafile (exclusively windows)

SVG : Scalable vector graphics (open standard)

DXF : for technical drawings (Autocad)

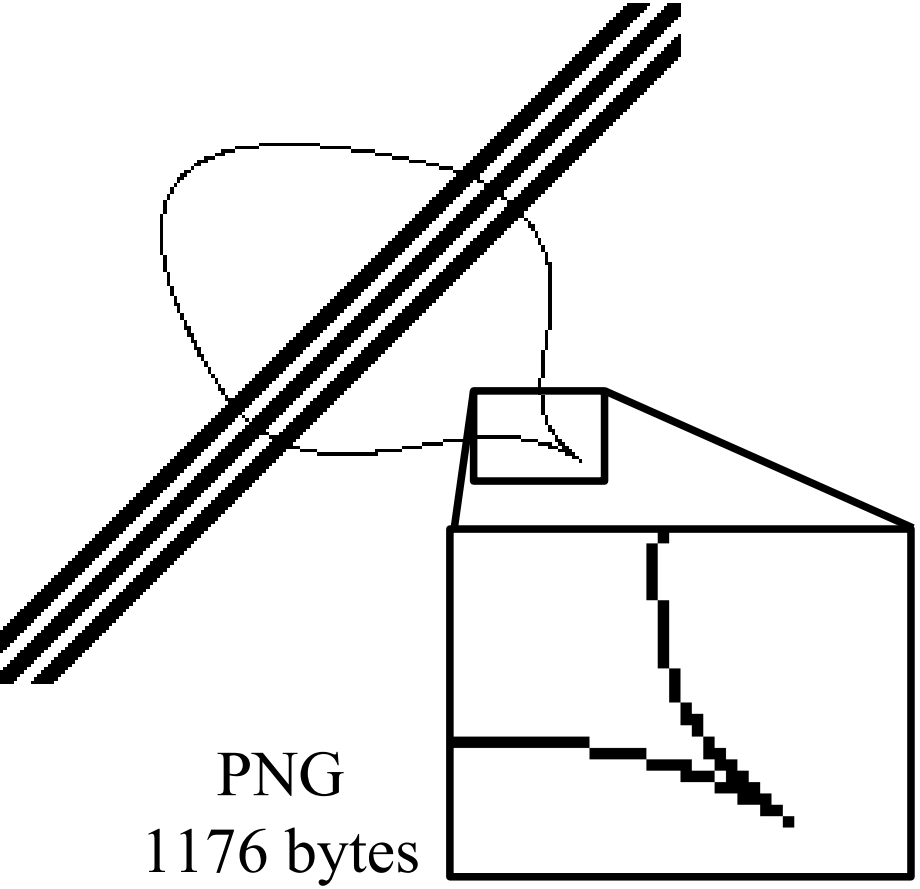
+ proprietary formats : coreldraw, adobe illustrator...

Image storage

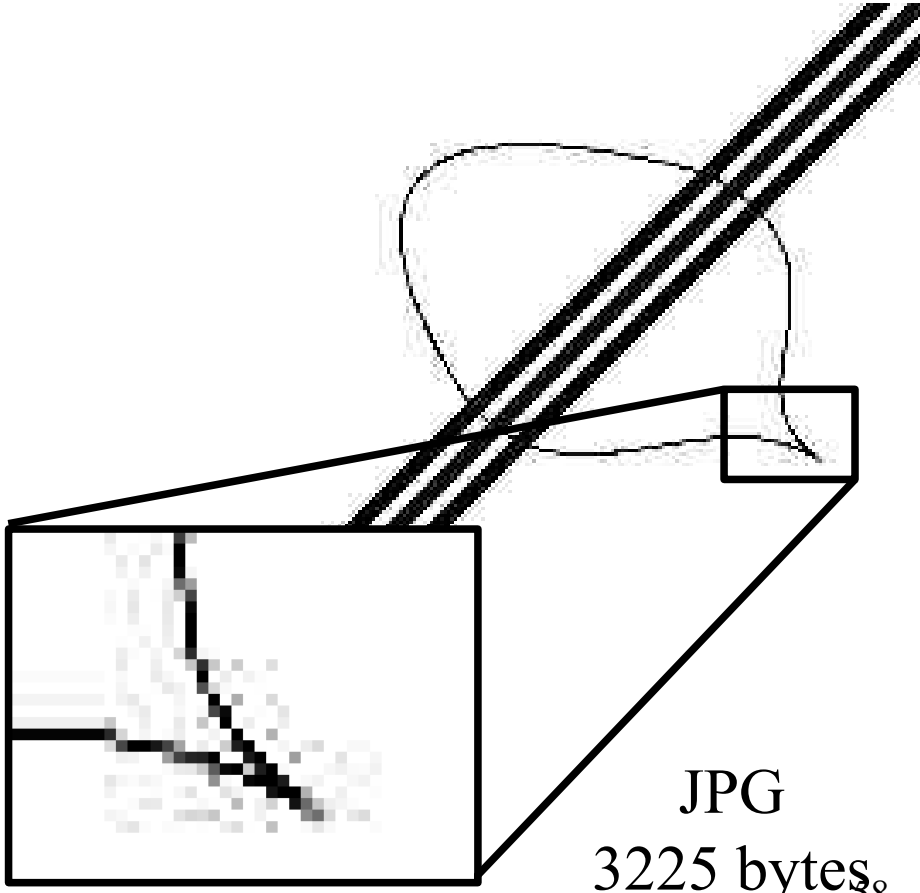
- Bitmap images
  - Uncompressed storage
    - BMP (old), TIFF (1/8/16 bits/c, floating point) , PNG (8,16 bits/c, alpha channel,1,2,4,8 bits indexed col.), PNG (indexed colors + alpha channel), EXR (floating point 16,32 bits/channel)
  - Lossless compression
    - TIFF, PNG, GIF, EXR
  - Lossy compression
    - TIFF, JPG(8 bits), JPEG2000 (improved JPG but not used due to patents !)

Image storage

- For images with few colours, and/or with sharp contrasts, never use JPG.



PNG
1176 bytes



JPG
3225 bytes

Image storage

- Pictures or images with continuous tones are adapted to JPG.

24 bit JPG : 85Kbytes

Image storage

- Pictures or images with continuous tones are adapted to JPG.

24 bit PNG / lossless : 646Kbytes

Image storage

- Pictures or images with continuous tones are adapted to JPG.

8 bit PNG (256 indexed colours) : 227 Kbytes

Image storage

- Pictures or images with continuous tones are adapted to JPG.

4 bit - PNG (16 indexed colours) : 108 Kbytes

Image storage

- High contrast images (HDR) or images that will be manipulated (brightness/contrast ...)
  - 16 bit PNG
  - 16 bit TIFF
  - EXR
- Images for the WWW or for display on desktop screen
  - GIF, 8 bit PNG, JPG
- If space is not a problem, always prefer lossless compression and a high nb of bit/channel.

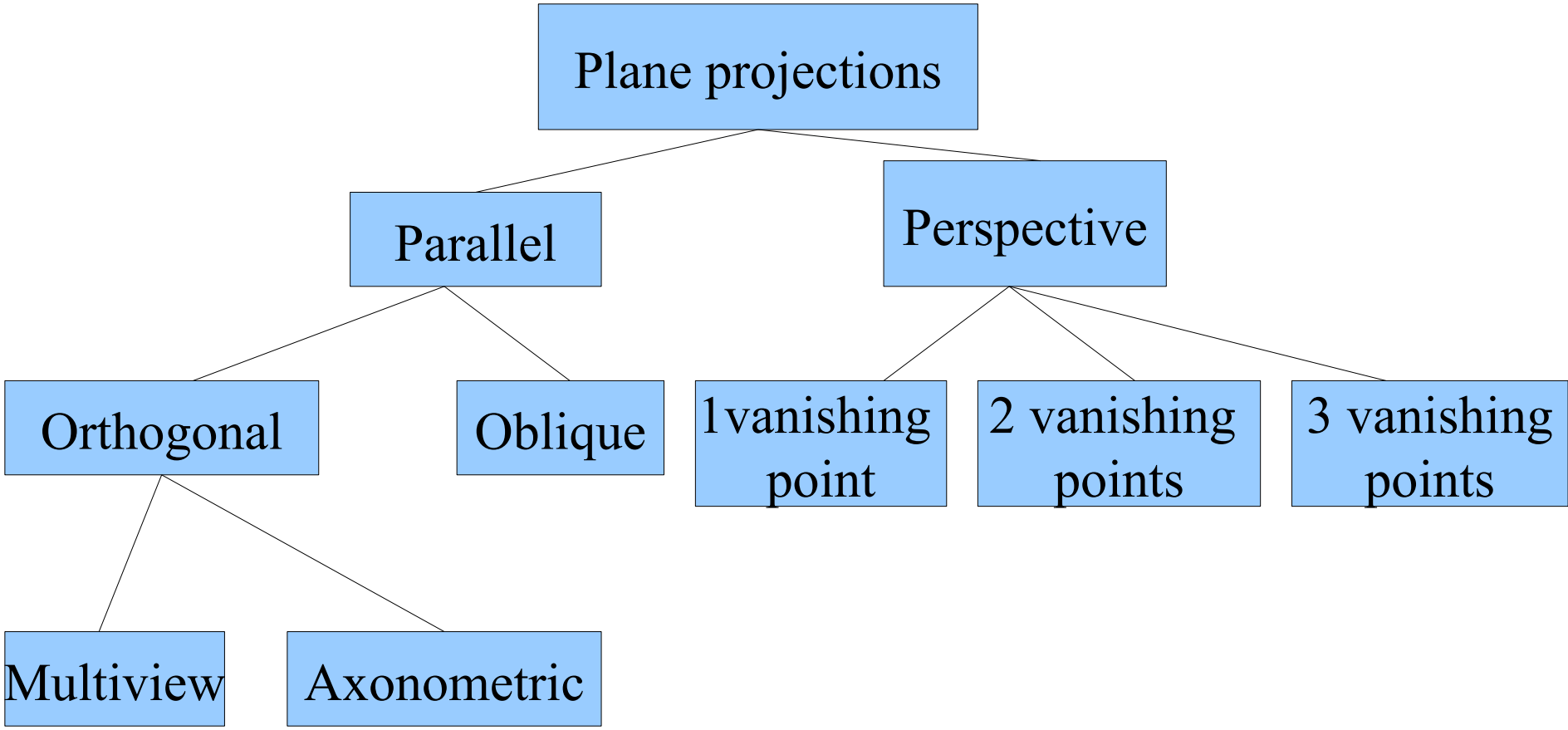
Image storage

There is no universal image format !

Perspective and transformation matrices

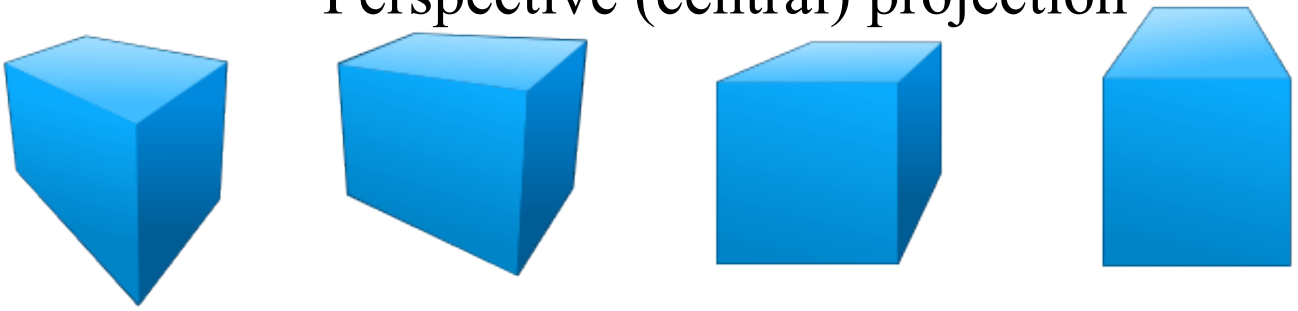
Perspective

- « Classical » projections

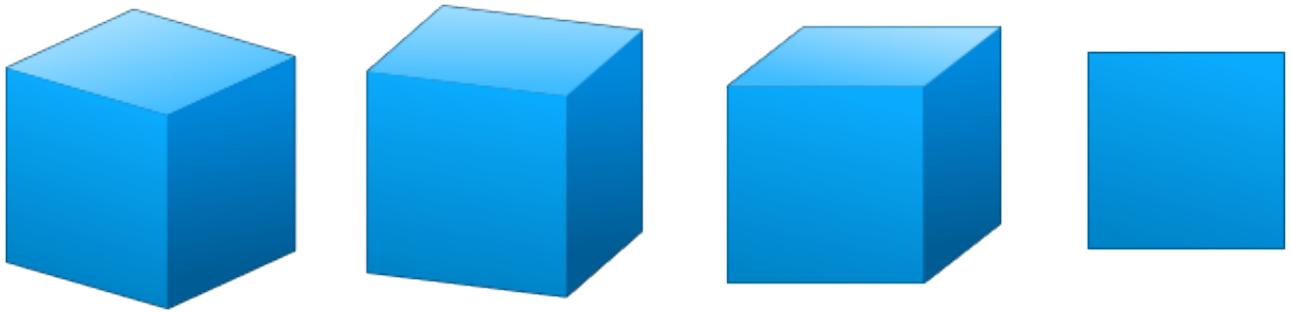


Perspective

Perspective (central) projection



Parallel projection



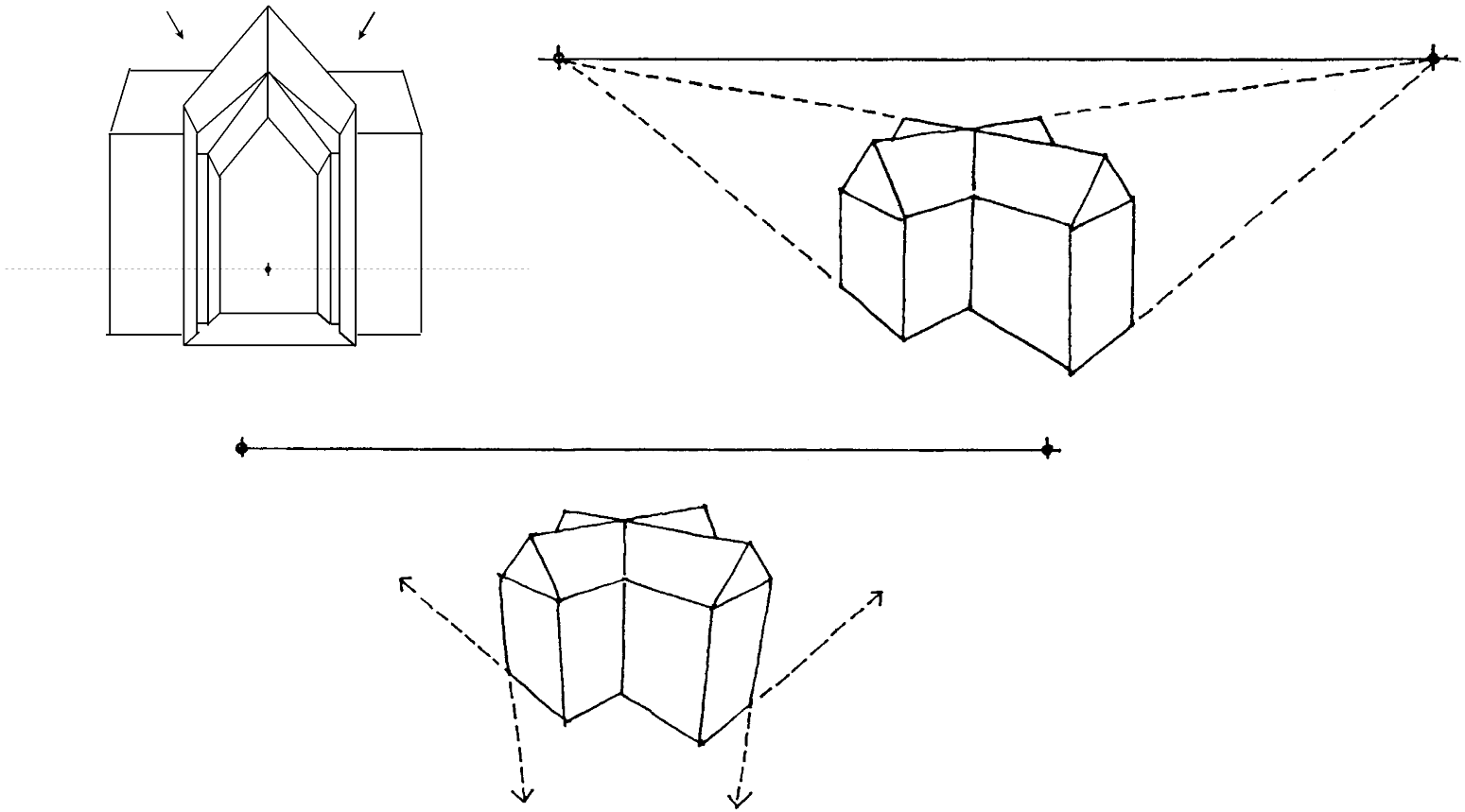
Oblique projection

Axonometric

Orthogonal

Perspective

- Vanishing points in perspective projection

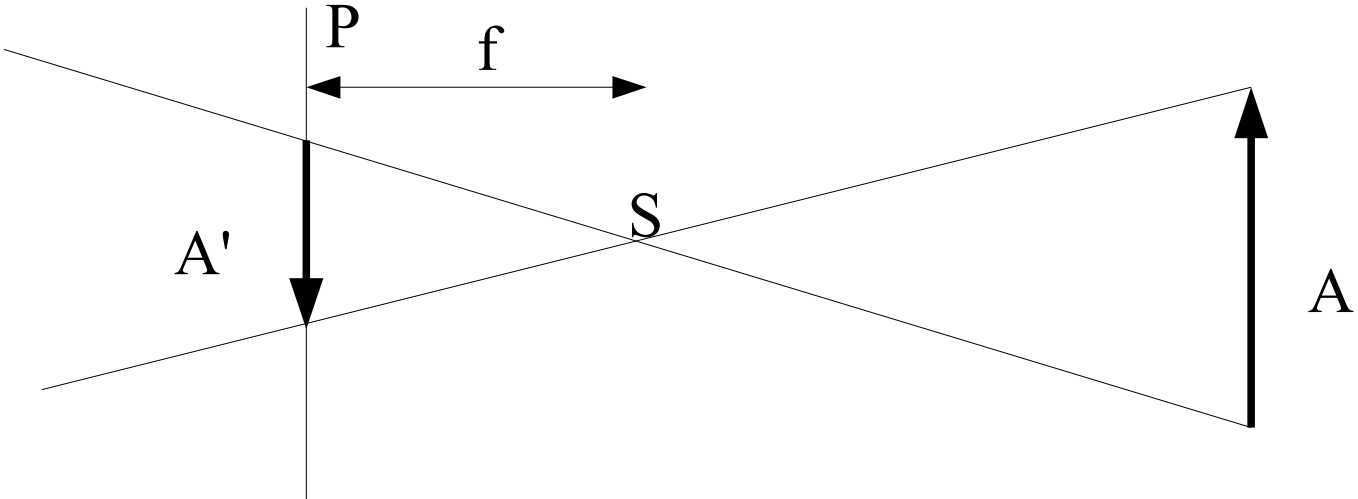


Computer Graphics

Perspective

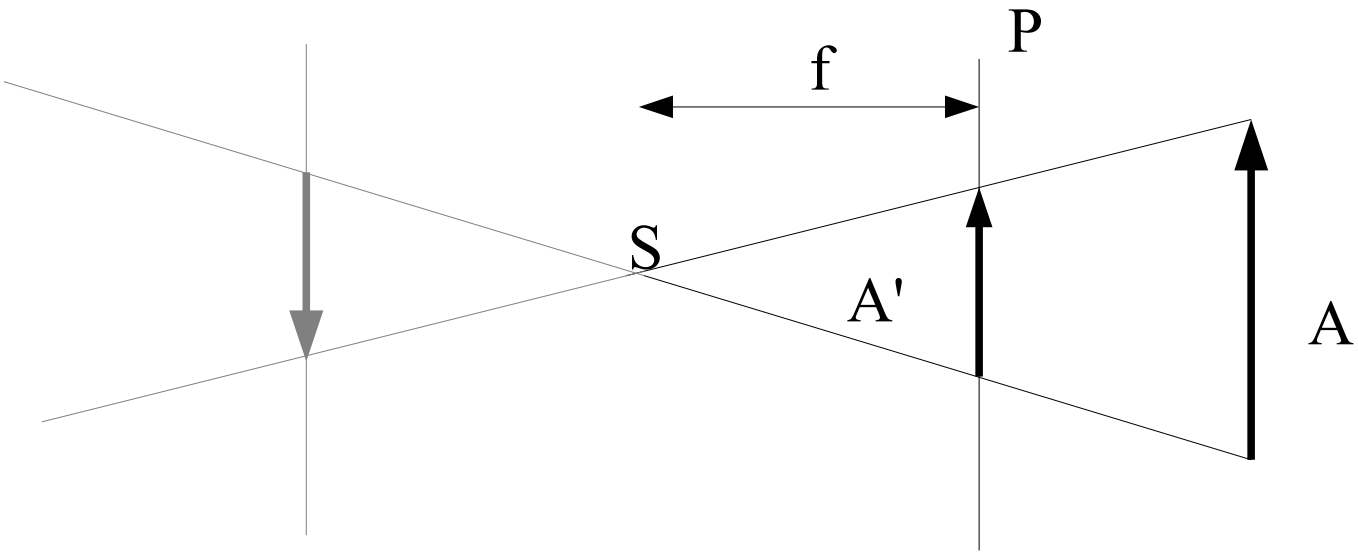
Perspective

- The perspective is a fairly accurate representation of what the eye sees
  - Based on central projection
  - In first approximation, the eye (or a camera) is made of a lens (the eye's lens), and a projection plane of the image (retina). The lens may be considered as a point in what follows.



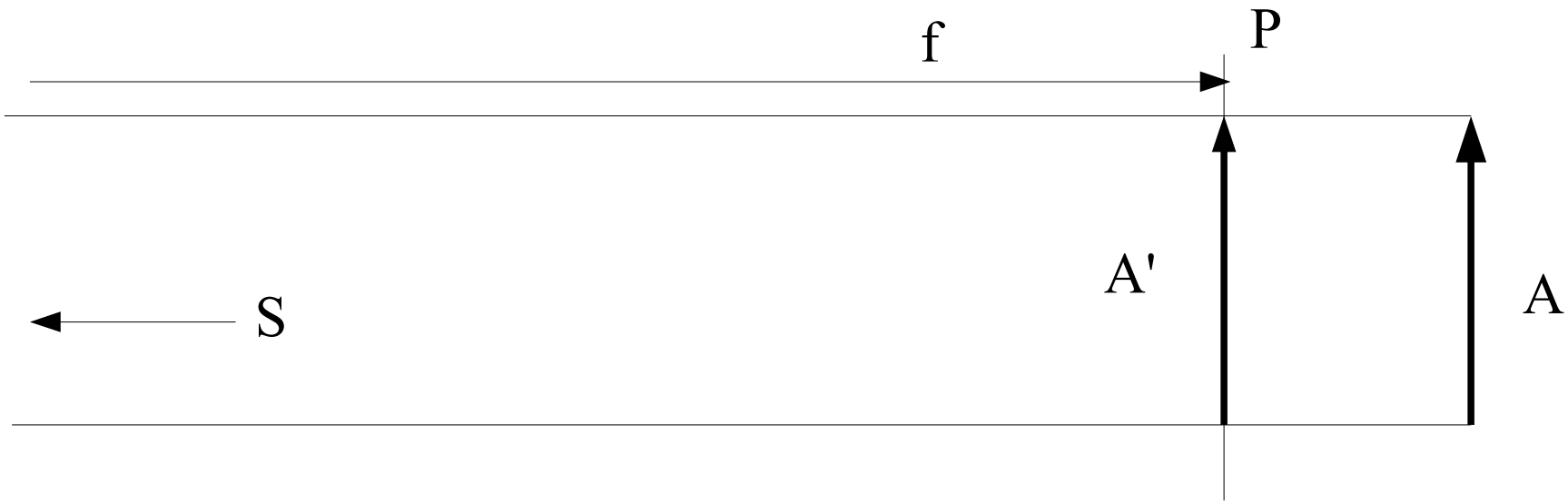
Perspective

- Equivalent configuration used in computer graphics



Perspective

- Parallel projection is the limiting case where f tends to infinity.



Geometric transformations

Transformation matrices

- Geometric transformations
  - Two goals
    - Get from 3D coordinates objects a projection on the screen plane (coordinates 2D + depth details)
    - From elementary objects, they can be placed anywhere in the volume, optionally modified by operations such as shearing or scaling.

Affine transformations

- Case of linear transformations
  - Affine transformations

$$\phi(\boldsymbol{P}) \equiv \mathbf{A} \cdot \boldsymbol{P} + \boldsymbol{u}, \quad \boldsymbol{u} \in \mathbb{R}^3$$

Some affine transformations

identity : $\boldsymbol{u} = 0$, $\mathbf{A} = \mathbf{I}$, $\mathbf{I}$ is the identity matrice,

$$\boldsymbol{u} = 0 \;;\; \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

translation :
$\boldsymbol{u}$ is the translation vector, $\mathbf{A} = \mathbf{I}$,

$$\boldsymbol{u} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \;;\; \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

scaling
$\boldsymbol{u} = 0$, $\mathbf{A}$ is a diagonal matrice, whose terms define the scales along the axes,

$$\boldsymbol{u} = 0 \;;\; \mathbf{A} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}$$

rotation :
$\boldsymbol{u} = 0$, $\mathbf{A}$ is the rotation matrice,

$$\boldsymbol{u} = 0 \;;\; \mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Some affine transformations

shearing :
where a, b, c are the 3shearing coefficients.

$$\boldsymbol{u}=0 \;\; ; \;\; \mathbf{A}=\begin{bmatrix} 1 & a & b \\ 0 & 1 & c \\ 0 & 0 & 1 \end{bmatrix}$$

Important particular case :
If the matrice **A** is orthogonal : $\mathbf{A}^T = \mathbf{A}^{-1}$

then this transformation preserves angles and lengths.

Matrix treatment

Multiplicative treatment

$$P_1 = \mathbf{S} \cdot P_0$$
$$P_1 = \mathbf{R} \cdot P_0$$
$$P_1 = \mathbf{C} \cdot P_0$$

Additive treatment for the translation

$$P_1 = P_0 + t$$

The treatment is not the same for all operations ...

Homogeneous coordinates

- In order to make the identical treatment of the translation, we add one coordinate, set to 1 for now
- The additional coordinate will be used for the perspective projection in the sequel

$$\begin{bmatrix} 1 & 0 & 0 & r \\ 0 & 1 & 0 & s \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+r \\ y+s \\ z+t \\ 1 \end{bmatrix}$$

Other operations

Translation

Transformation matrices

- translation : $\mathbf{D}(\boldsymbol{t}) = \begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; \quad \boldsymbol{t} = \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$

- scaling : $\mathbf{S}(p, q, r) = \begin{bmatrix} p & 0 & 0 & 0 \\ 0 & q & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Transformation matrices

- rotation around z-axis : $\mathbf{R}_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

- rotation around x-axis : $\mathbf{R}_x(\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -\sin\theta & 0\\ 0 & \sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

- rotation around y-axis : $\mathbf{R}_y(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta & 0\\ 0 & 1 & 0 & 0\\ -\sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

Transformation matrices

- shearing (general case)

$$\mathbf{C}(a,b,c)=\begin{bmatrix}1 & a & b & 0\\ 0 & 1 & c & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

- One can combine these transformation matrices by simple multiplication
  - One should respect the ordering (not commutative !)

$$\mathbf{G}_n=\mathbf{G}_{n-1}\cdots\mathbf{G}_2\cdot\mathbf{G}_1$$

Transformation matrices

- In particular, if one expresses an additional transformation T:
  - Relative to the origin of the reference (O)

$$\mathbf{G}_n = \mathbf{T} \cdot \mathbf{G}_{n-1}$$

  - Relative to the object (transform of O by $\mathbf{G}_{n-1}$)

$$\mathbf{G}_n = \mathbf{G}_{n-1} \cdot \mathbf{T}$$

  - Relative to an arbitrary point A

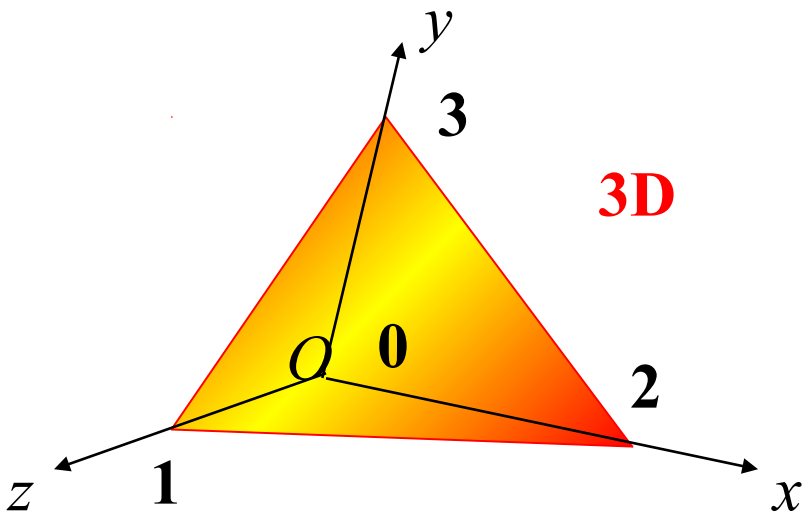
$$\mathbf{G}_n = \mathbf{D}(x_A, y_A, z_A) \cdot \mathbf{T} \cdot \mathbf{D}(-x_A, -y_A, -z_A) \cdot \mathbf{G}_{n-1}$$

Transformation matrices

- Change of reference (for parallel projections)
  - 3D global reference to reference 2D screen
  - 12 parameters in the transformation
  - We can specify the transformation uniquely by 4 points (forming a tetrahedron) and their transforms

Transformation matrices

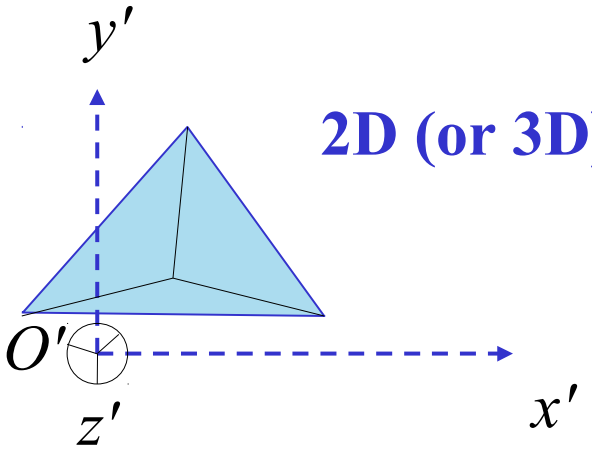
$$P_0 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



$$P_1 = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P_1 = \mathbf{T} \cdot P_0$$

$$\mathbf{T} = P_1 \cdot P_0^{-1}$$



Transformation matrices

$$\boldsymbol{P}_0 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \boldsymbol{P}_0^{-1} = \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{P}_1 = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 & x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 & y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix is used to project the coordinates in an arbitrary axis system

Transformation matrices

- Using the transformations that we have just seen , we can :

  - Transform space coordinates (x, y, z) coordinates to screen coordinates (x ', y', z '= depth), expressing the position in space of the view associated with the screen

  - Consider a scale factor to adjust the size of the virtual screen space and switch from (x', y') to (i, j) which is the geometrical position on the screen

  - The third coordinate (depth z') will be used to compute hidden faces.

Transformation matrices

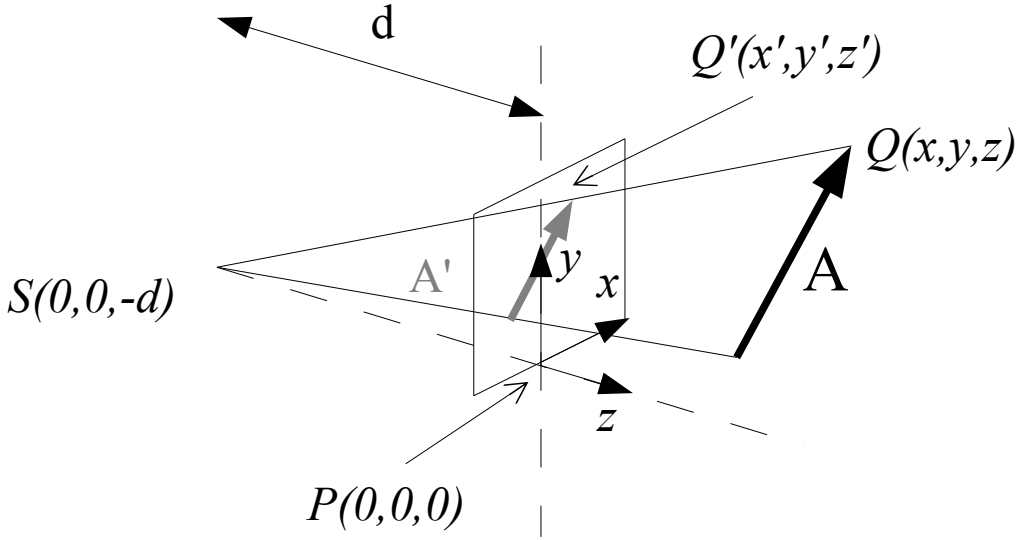
- For points where the 4th (homogenous) coordinate w is different from one
  - We consider that all points situated along a line going through the origin are **equivalent**
  - This corresponds to a central projection on the hyperplane $w = 1$; the following points are (geomtrically) equivalent:

$$[wx, wy, wz, w] \Leftrightarrow [x, y, z, 1]$$

  - If $w = 0$, it means a vector in homogeneous coordinates $[x, y, z, 0]$
  - There is a way to distinguish vectors and points !

Transformation matrices

- Perspective transformation
  - We will consider a special case that does not betray the generality of the approach
    - P is at the origin O of the reference frame
    - The screen is a plane of normal $n=(0,0,1)$ (perpendicular to z), containing P
    - We look towards the positive z.



$$\frac{x'}{x}=\frac{y'}{y}=\frac{d}{(d+z)}$$

$$z'=0$$

Transformation matrices

- Screen coordinates in function of space coordinates

$$x' = \frac{x}{1+\frac{z}{d}} \qquad y' = \frac{y}{1+\frac{z}{d}} \qquad z' = 0$$

  - We will change that so that the 3rd coordinate is going through the same scaling

$$x' = \frac{x}{1+\frac{z}{d}} \qquad y' = \frac{y}{1+\frac{z}{d}} \qquad z' = \frac{z}{1+\frac{z}{d}}$$

Transformation matrices

- The three Cartesian coordinates

$$x' = \frac{x}{1+\frac{z}{d}} \qquad y' = \frac{y}{1+\frac{z}{d}} \qquad z' = \frac{z}{1+\frac{z}{d}}$$

may be expressed differently in the homogeneous space (via the equivalence relation) :

$$\left(\frac{x}{1+\frac{z}{d}}, \frac{y}{1+\frac{z}{d}}, \frac{z}{1+\frac{z}{d}}, 1 \right) \equiv (x, y, z, 1+\frac{z}{d})$$

Transformation matrices

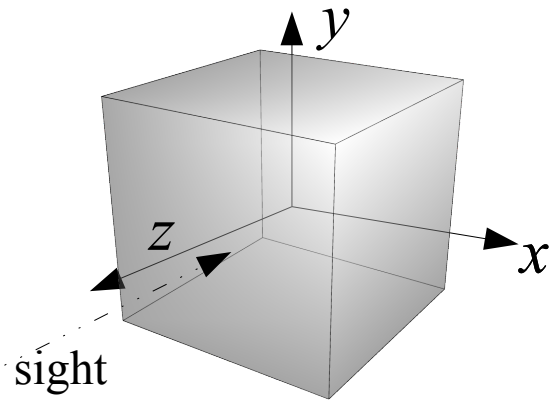
- We can then write the transformation as a **linear** transformation (in fact, a shear) in the homogeneous space

$$\begin{pmatrix} x \\ y \\ z \\ 1+\frac{z}{d} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

  - The following operations lead to the desired image :
    - First, the 4D “shearing”
    - Then, a perspective projection into 3D space, it is called **the perspective division** - it is a **non-linear** transformation
    - Then an orthogonal projection on the screen for which z = constant

Transformation matrices

- Field of view and effective construction of transformation matrices
  - Canonical view field
    - $(x_c, y_c, z_c) \in [-1,1]^3$
    - Screen : consists in (n_x, n_y) pixels, centred at the origin, in the plane $(x_p, y_p) \in [-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$
    - Transformation of the canonical field to the screen coordinates



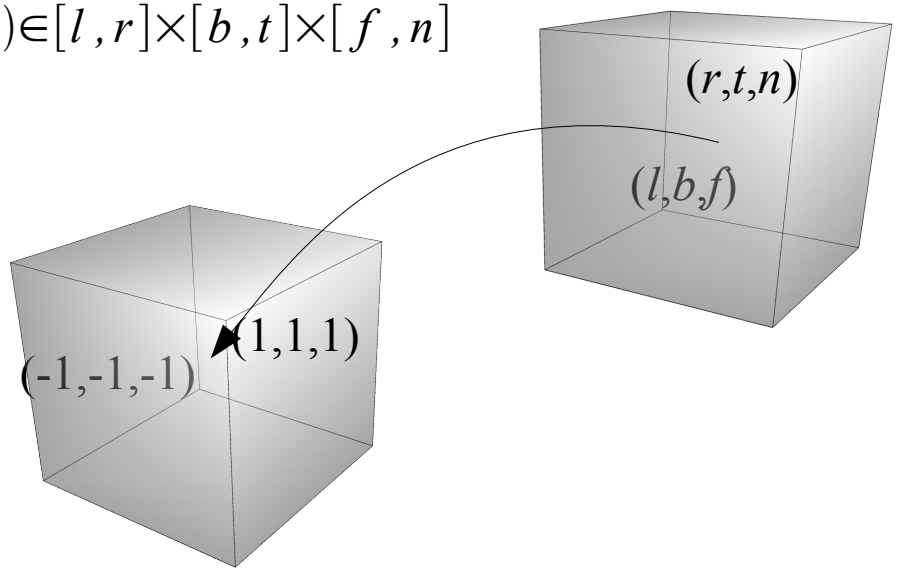
$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & -\frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M_s} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}$$

NB : If y_c is reversed, it must be taken into account ...

Transformation matrices

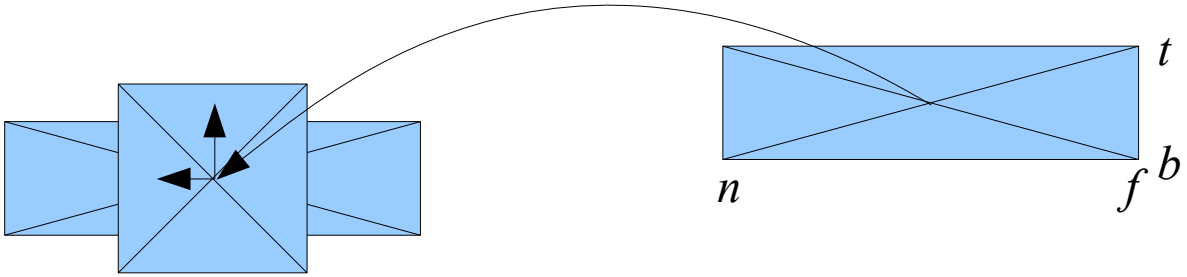
- Orthographic projection
  - We will match a specified volume with the canonical volume
    - This volume is aligned with the canonical volume but does not have the same center, nor the same dimensions

$$(x_d, y_d, z_d) \in [l, r] \times [b, t] \times [f, n]$$



Transformation matrices

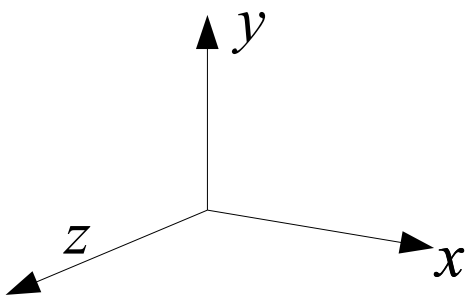
- The process resorts to a translation followed by a dilation.

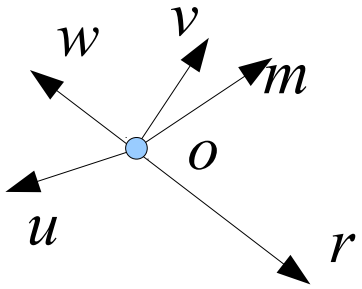


$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M_c} \cdot \begin{pmatrix} x_d \\ y_d \\ z_d \\ 1 \end{pmatrix}$$

Transformation matrices

- Viewpoints with an arbitrary position and orientation
  - One would like to watch in an arbitrary direction and from any point
    - The position of the eye (o), the view direction (r) and “noon” -local vertical line for the observer- (m) are defined
    - An orthonormal (o, u, v, w) frame is constructed from the data.





$$w = -\frac{r}{\|r\|}$$

$$u = -\frac{m \times w}{\|m \times w\|}$$

$$v = w \times u$$

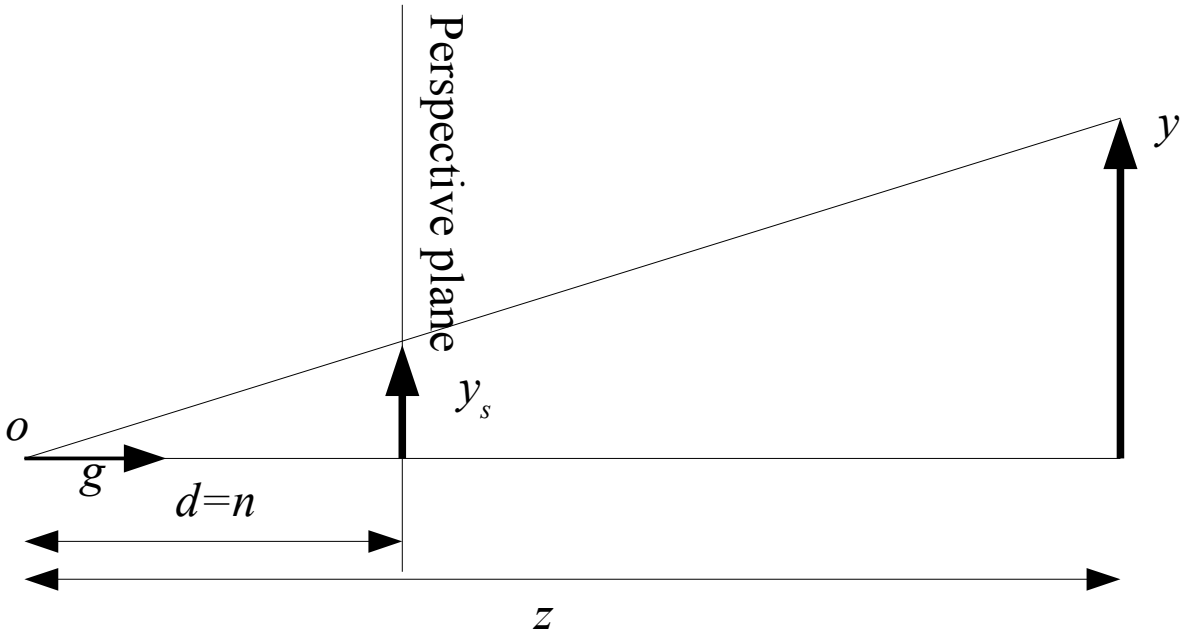
Transformation matrices

- The alignment of the space coordinates with those of the viewer is done with two changes
  - Translation bringing the coordinates of the eye to the origin
  - Rotation about the axes to align them with the global axes

$$\begin{pmatrix} x_d \\ y_d \\ z_d \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_o \\ 0 & 1 & 0 & -y_o \\ 0 & 0 & 1 & -z_o \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M_v} \cdot \begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix}$$

Transformation matrices

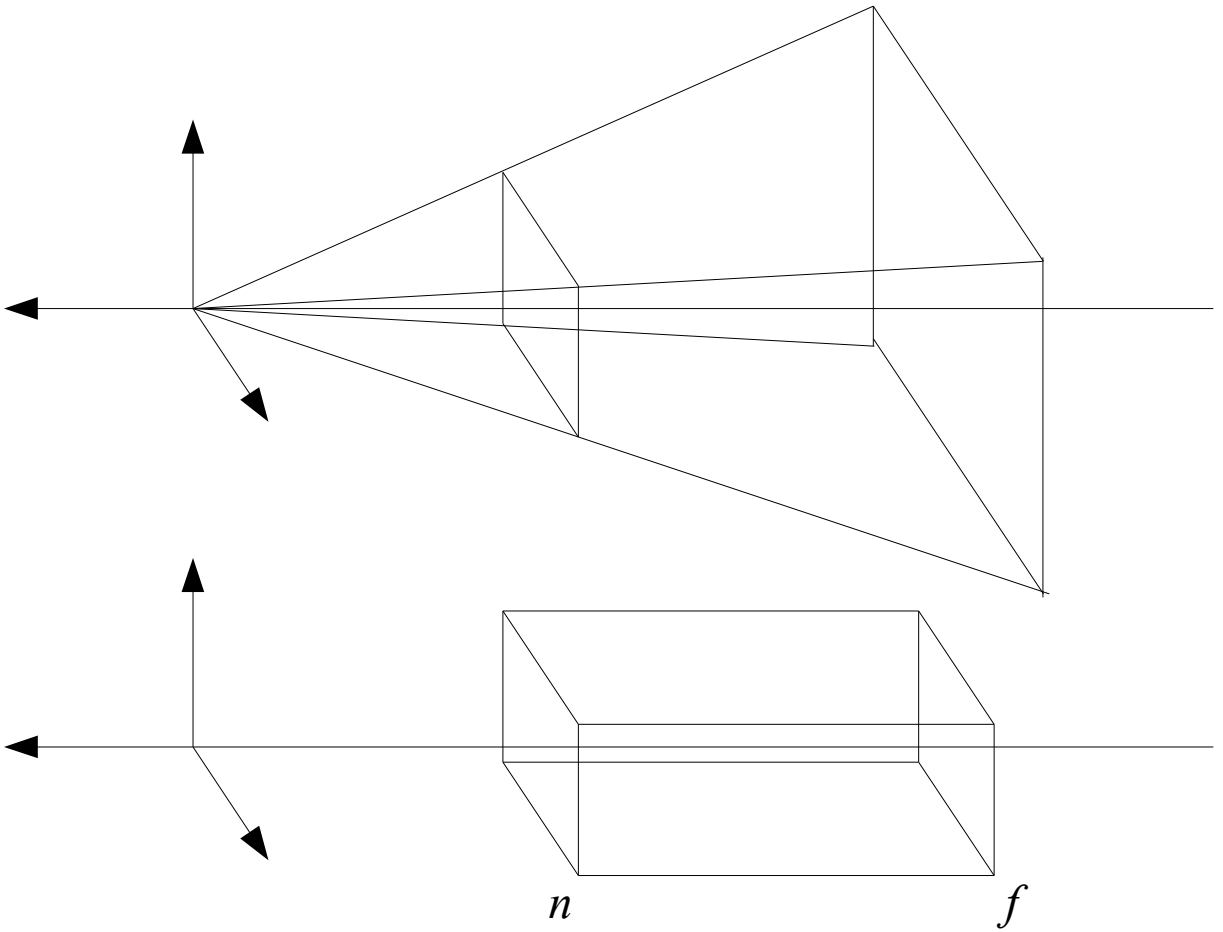
- Perspective transformation



$$y_s = \frac{d}{z} y$$

Transformation matrices

- Here we want to keep the 'z' value of plane f and keep (x, y, z) on plane n (projection plane or perspective plane)



Transformation matrices

- To use homogeneous coordinates,
  - All three components must be divided by the same value
  - Recall what we saw before :

$$\begin{pmatrix} x \\ y \\ z \\ 1+\frac{z}{d} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

    - Compared to this matrix, it has a displacement to bring the eye to $(0,0,0)$, thus keeping z=n unchanged
    - And a small change to keep the points $z = f$ unchanged as well

Transformation matrices

- Keep $z = f$ and $z = n$ unchanged

$$\begin{pmatrix} x\frac{n}{z} \\ y\frac{n}{z} \\ (z\alpha+\beta)\frac{n}{z} \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & \frac{1}{n} & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\begin{cases} n\frac{(n\alpha+\beta)}{n} = n \\ n\frac{(f\alpha+\beta)}{f} = f \end{cases} \Rightarrow \begin{cases} \alpha = \frac{n+f}{n} \\ \beta = -f \end{cases}$$

Transformation matrices

- Perspective matrix

$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & -f \\ 0 & 0 & \frac{1}{n} & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

  - The matrix can be multiplied by an arbitrary constant (because of the homogeneous coordinates). Multiplying by n yields :

$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix} \equiv \underbrace{\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -f\,n \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{M_p} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Transformation matrices

- The perspective matrix that we have just defined suppose one looks in the direction of the negative z
  - We must therefore apply it **after** the change of point of view !
  - The complete chain of transformations is therefore :

$$M = M_s \cdot M_c \cdot M_p \cdot M_v$$

Transformation matrices

- In particular, the matrix

$$M_{proj\_persp} = M_c \cdot M_p = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- is called perspective projection matrix, and allows to reach from the real space the canonical volume $[-1,1]^3$

$$M = M_s \cdot M_{proj\_persp} \cdot M_v$$

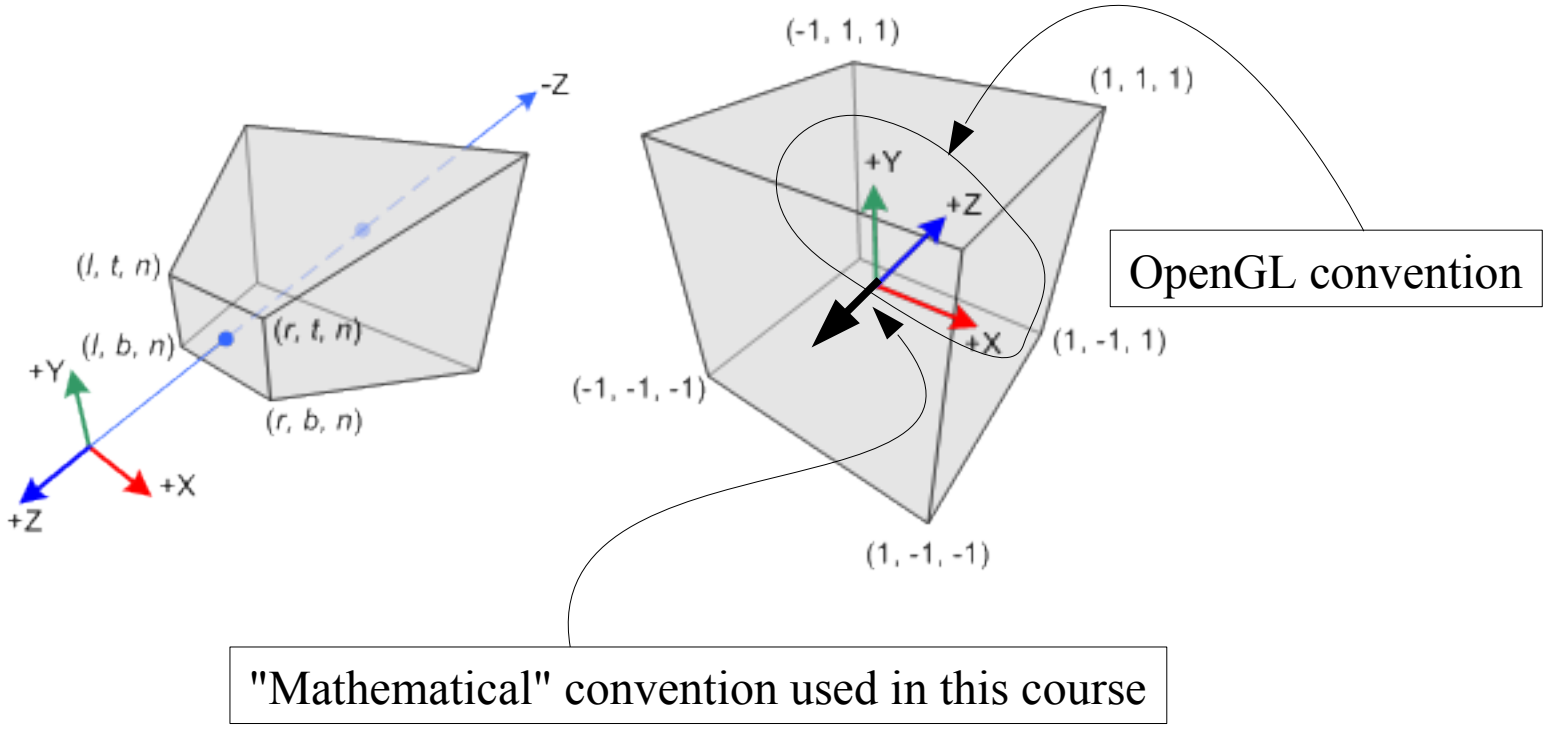
$$z = -|n| \rightarrow z_c = 1$$

$$z = -|f| \rightarrow z_c = -1$$

Depends on the hardware	Depends on the type of projection	Depends on your point of view

Transformation matrices

- OpenGL...



Transformation matrices

- By multiplying by -1 and substitute

$$f<n<0 \quad \text{et} \quad -n=|n| \quad -f=|f| \quad nf=|n||f| \quad n-f=|f|-|n|$$

one obtains :

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix} \equiv \begin{pmatrix} \frac{-2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{-2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-f-n}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix} \equiv \begin{pmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|f|+|n|}{|f|-|n|} & \frac{2|f||n|}{|f|-|n|} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Transformation matrices

- Conventional OpenGL perspective projection matrix

$$\begin{pmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|f|+|n|}{|f|-|n|} & \frac{2|f||n|}{|f|-|n|} \\ 0 & 0 & -1 & 0 \end{pmatrix} \rightarrow \underbrace{\begin{pmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|f|+|n|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{pmatrix}}_{M_{proj\_persp\_OpenGL}}$$

$$z=-|f| \rightarrow z_c=1$$

$$z=-|n| \rightarrow z_c=-1$$

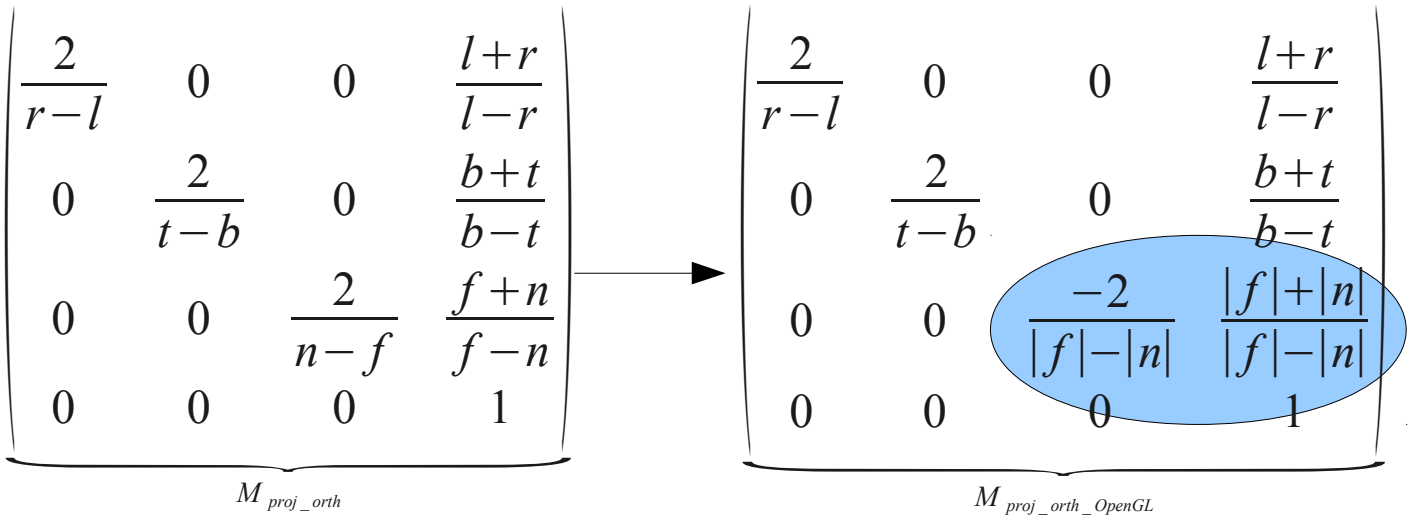
Transformation matrices

- Idem without perspective transformation

$$\underbrace{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M_c} = \underbrace{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{M_{proj\_orth}}$$

Transformation matrices

- OpenGL orthographic projection matrix



- In all cases :

$$M = M_s \cdot M_{proj} \cdot M_v$$

Transformation matrices

- Usual simplifications
  - In general, one looks at the center of the volume

$$[l,r]\times[b,t]\times[f,n]$$

$$r=-l=\frac{w}{2} \qquad t=-b=\frac{h}{2}$$

  - We also want square pixels

$$\frac{w}{h}=\frac{n_x}{n_y}$$

$$M_{proj\_orth\_OpenGL}=\begin{pmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{n_x}{w\,n_y} & 0 & 0 \\ 0 & 0 & \frac{-2}{|f|-|n|} & \frac{|f|+|n|}{|f|-|n|} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{proj\_persp\_OpenGL}=\begin{pmatrix} \frac{|n|}{w} & 0 & 0 & 0 \\ 0 & \frac{|n|\,n_x}{w\,n_y} & 0 & 0 \\ 0 & 0 & \frac{|f|+|n|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Transformation matrices

- One can also specify the field of view (obviously for perspective projections only)

$$\tan\theta = \frac{w}{|n|} \Rightarrow w = |n|\tan\theta$$

$$M_{proj\_persp\_OpenGL} = \begin{pmatrix} \frac{1}{\tan\theta} & 0 & 0 & 0 \\ 0 & \frac{n_x}{n_y \tan\theta} & 0 & 0 \\ 0 & 0 & \frac{|f|+|n|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Transformation matrices

- Exercise
  - Compute the transformation matrix for an observer located at (x = 10, y = 10, z = 10) facing the direction (-1, -1, -1). A vector from the vertical plane is the vector (0,1,0). The screen is 1000 by 1000 pixels.
  - Angle of view : 45° (tan 45 = 1)
  - Plane position n and f : z=10 and z=20 respectively

$$w = -\frac{g}{\|g\|} \qquad u = -\frac{t \times w}{\|t \times w\|}$$

$$v = w \times u$$

$$M_s = \begin{pmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{proj\ persp\ OpenGL} = \begin{pmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|f|+|n|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$M_v = \begin{pmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_o \\ 0 & 1 & 0 & -y_o \\ 0 & 0 & 1 & -z_o \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformation matrices

- How to select (point at an object with the mouse and pick the object)
  - Using the reverse transformation
  - Need to know for each pixel, the geometric primitive recently drawn
    - We'll see how later...

Two paradigms for synthetic image generation

Computer Graphics

2 paradigms...

- A projection of the objects on the plane of the screen
  - Purely geometrical aspects
    - Using transformation matrices
    - Need for hidden line removal algorithm
    - « Clipping » and « culling » techniques
      - Allows to draw only visible entities (and minimizing side effects)
  - Colouring / shadowing
    - Lighting
    - Textures
    - Laws of reflection
  - Possibility of real-time graphics (e.g. videogames, ...)
  - OpenGL type implementation (in hardware)

2 paradigms...

- Projective “à la OpenGL” paradigm
  - Start from the objects and their coordinates in space
  - Determine at each point or every facet, lighting features, textures, etc ...
  - Project into the coordinate space of the screen
    - Transformation matrices seen before
  - Draw the object in discrete form
    - Raster algorithm - (!) aliasing
    - hidden faces !
    - The colour of each pixel is determined by the information calculated above (previously interpolated or computed on-the-fly)

Computer Graphics

2 paradigms...

- Ray-tracing paradigm
  - Geometric aspects
    - We start form pixels to meet objects
      - Finding the intersection of a ray and objects (usually triangulated)
    - Problem of the hidden faces : automatically solved !
    - Culling is useful (to avoid unnecessary searches)
  - Colouring
    - Realistic reflection laws
    - Realistic lighting law
    - Complex textures
  - Relatively slow but higher fidelity
  - Easy introduction of new light/texture models

2 paradigms...

- Ray-tracing paradigm
  - Start from screen pixels
  - Draw a line through the eye and that pixel and calculate the intersection with the first object encountered online
    - Particular case if the surface is a dielectric or metal (smooth metal or glass, liquid etc ...) : the ray is reflected, or deflected or both
  - Determine the color of the object at this point
    - Use of physical or empirical laws
    - Consider the light - and shadows
  - Give the determined pixel color
    - (!) again, problems of potential aliasing !

2 paradigms...

- Common issues for both techniques
  - Laws of light-surface interaction, reflection, types of surfaces
  - Textures
  - Geometrical modelling
  - Meshes
  - Colour theory
  - Aliasing

2 paradigms...

- Specific topics to ray-tracing
  - Radiosity equation
  - Calculation of shadows
  - Refraction, reflection
  - Use of Computational Geometry to lower the computation costs

2 paradigms...

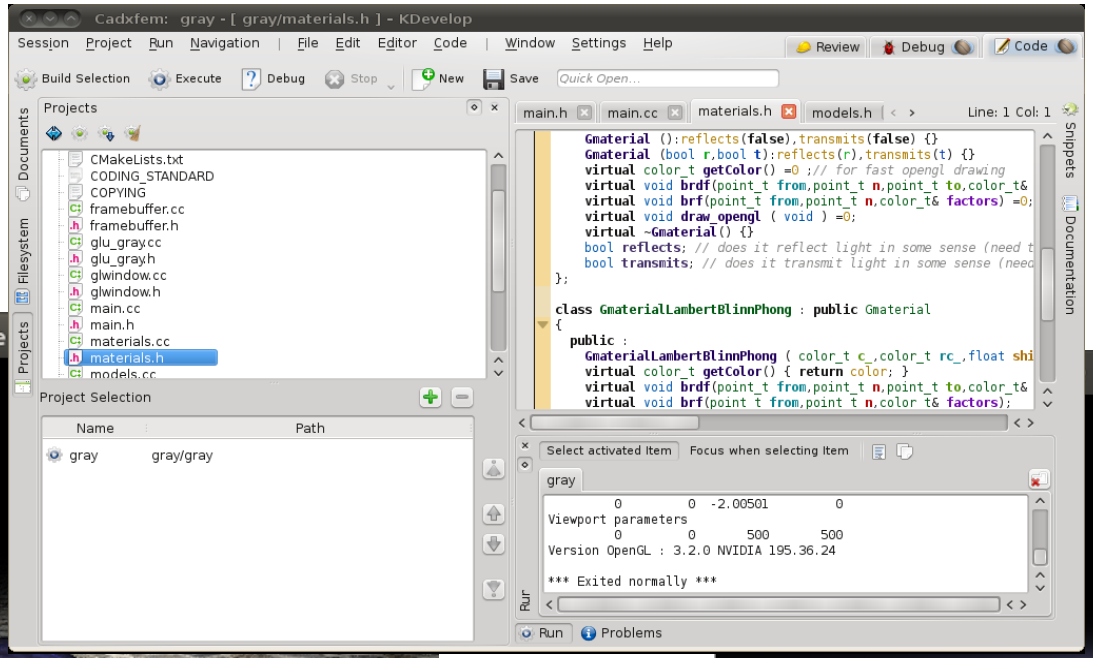
- Specific issues to the projective approach
  - Visibility problems (clipping and hidden faces)
  - Raster algorithms

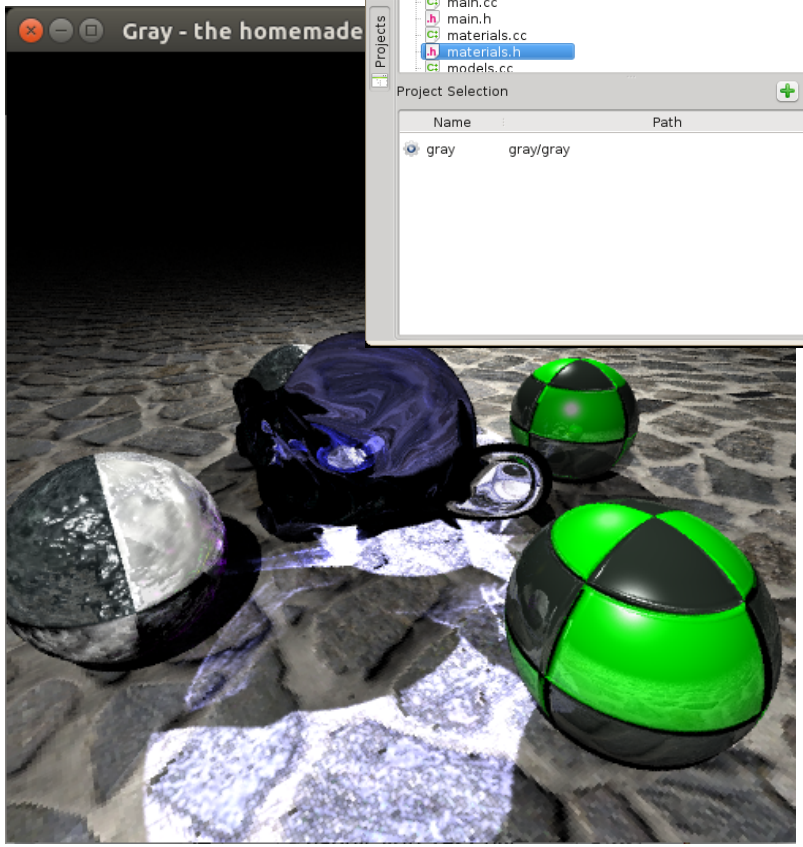
2 paradigms...

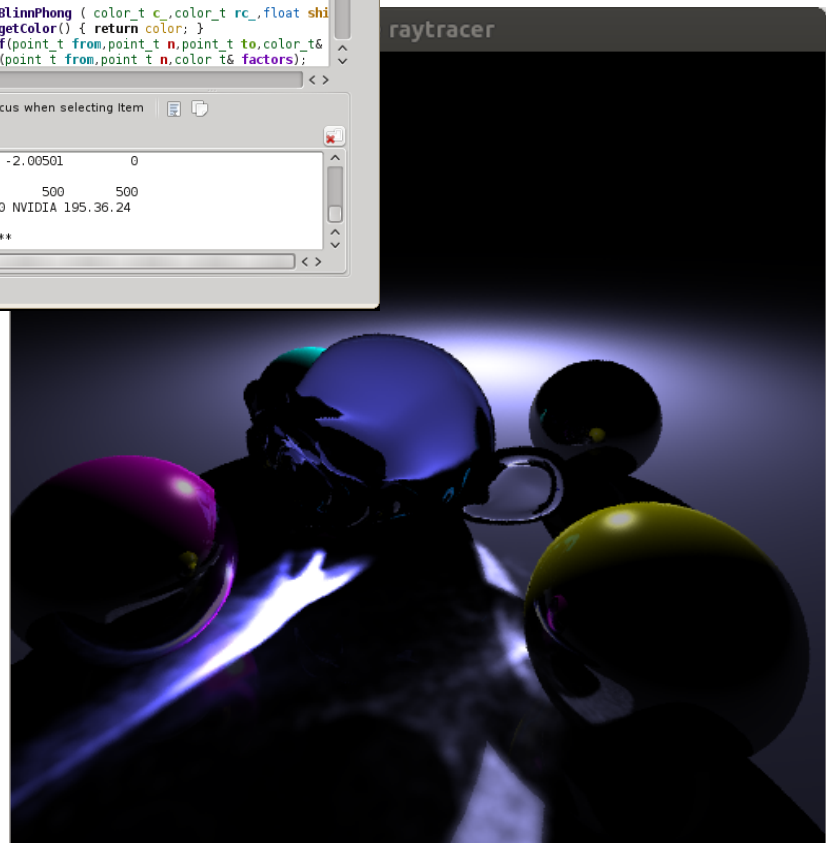
- There are hybrid approaches
  - Allows to combine the advantages of both paradigms
    - Speed (projective approach)
    - Fidelity (ray tracing approach)
    - They are more or less close to a very versatile implementation in software of the more rigid OpenGL stack.
  - E.g. Pixar Renderman.

Computer Graphics

Project







Project

- Available topics
  - A) Homemade Ray Tracer
    - Upgrade an existing ray-tracing program (which I provide)
    - Programming & Implementation
  - B) Topic of your choice
    - If you want to work on a topic that interests you and is within the scope of the course (I have to validate early enough !)

Project

- Project guidlines
  - Individual project
  - Deliverables
    - (A) Homemade Ray Tracer
      - a) “Specifications” : definition of what you will do (+ who) , planning ...
      - b) Final report detailing the philosophy of your contribution (why and how) + code + results and critical analysis + detailed personal contributions if you are two
      - c) Small presentation to show the results

Project

- (B) Own topic
  - a) Definition of the topic (originality, interest, context ...)
  - b) “Specifications” : definition of what you will do (+ who) , planning ...
  - c) Final report detailing the philosophy of your contribution (why and how) + code + results and critical analysis + detailed personal contributions if you are two
  - d) Small presentation to show the results
- Physical limits of the documents
  - Specifications : 1-2 A4 pages
  - Final report : max 15 A4 pages
  - Definition of the subject (B) : 1 A4 page

Project

- Deadlines
  - Choice of subjects : Feb 27th
    - In the case C) definition of the subject : Feb 27th
  - Specifications and planning : March 13th
  - Submission of reports : before June 1st
  - Relative weight in the final mark : TBD

Project

- Homemade Ray-tracer
  - Canvas available on the course's website

    http://www.cgeo.ulg.ac.be/infographie
  - Developed in portable C ++ under Linux
  - Uses OpenGL and FLTK
    - I do not want you develop a significant GUI for what you're doing (takes too long) - simply implement models of reflection, shadows, geometry etc ... and test this directly in the code (in the main() function for example). GUI might be an option when everything else works.
    - There are (very) few comments ...
    - You are allowed to modify the existing code...

Project

- Subjects related to the HomeMade RayTracer
  - 1 – Implement subsurface scattering
    - Rendering of semi-transparent objects (e.g. skin)
  - 2 – Interfacing with an existing solid modeler
    - CATIA files, Solidworks, etc.
  - 3 – Metropolis algorithm
    - Unbiased physical rendering
  - 4 – Efficient ray intersection with smooth CAD surfaces
    - bézier/NURBS surfaces and the like
  - All – Gather all !
    - Keep in mind that you work together...part of the evaluation depends on the integration of all developments in the same source tree.